

Lecture Slides Cryptographic Protocols

Version 1.32
February 6, 2018

Berry Schoenmakers

Department of Mathematics and Computer Science,
Technical University of Eindhoven,

P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

1.a.m.schoenmakers@tue.nl
berry@win.tue.nl

Cryptographic Protocols (2DMI00)

www.win.tue.nl/~berry/2DMI00/

Spring 2018



Course Plan

Lecture notes consist of eight chapters:

- ① Introduction
- ② Key Exchange Protocols
- ③ Commitment Schemes
- ④ Identification Protocols
- ⑤ Zero-Knowledge Proofs
- ⑥ Threshold Cryptography
- ⑦ Secure Multiparty Computation
- ⑧ Blind Signatures

- Chapters 1–2 will be covered in Lectures 1–5.
- Chapter 3 will be covered in Lectures 5–6.
- Chapters 4–5 will be covered in Lectures 6–10.
- Chapters 6–8 will be covered in Lectures 10–13.

Course Grading 2DMI00

2DMI00 course: 5 ECTS

Written exam (incl. homework): 80%

MPyC programming assignment: 20%

Requirement: passing grade at least 5.5 for written exam.

Motivating Example: Secret Santa

Web server (acting as **trusted party**) generates random permutation π of $\{1, 2, \dots, n\}$ without fixed points (so $\pi(i) \neq i$).
Web server sends $\pi(i)$ to party P_i in private.

Cheating web server could

- pick an arbitrary function for π (e.g., with $\pi(1) = \pi(2)$)
- influence who picks who
- tell others who picked who
- ...

Q: How to do this by means of a cryptographic protocol—**no trusted party**?
A: Use a secure multiparty computation protocol!

Motivating Example: Secret Santa

Web server (acting as **trusted party**) generates random permutation π of $\{1, 2, \dots, n\}$ without fixed points (so $\pi(i) \neq i$).
Web server sends $\pi(i)$ to party P_i in private.

Cheating web server could

- pick an arbitrary function for π (e.g., with $\pi(1) = \pi(2)$)
- influence who picks who
- tell others who picked who
- ...

Q: How to do this by means of a cryptographic protocol—**no trusted party**?

A: Use a secure multiparty computation protocol!

Motivating Example: Secret Santa

Web server (acting as **trusted party**) generates random permutation π of $\{1, 2, \dots, n\}$ without fixed points (so $\pi(i) \neq i$).
Web server sends $\pi(i)$ to party P_i in private.

Cheating web server could

- pick an arbitrary function for π (e.g., with $\pi(1) = \pi(2)$)
- influence who picks who
- tell others who picked who
- ...

Q: How to do this by means of a cryptographic protocol—**no trusted party**?
A: Use a secure multiparty computation protocol!

Motivating Example: Secret Santa

Web server (acting as **trusted party**) generates random permutation π of $\{1, 2, \dots, n\}$ without fixed points (so $\pi(i) \neq i$).
Web server sends $\pi(i)$ to party P_i in private.

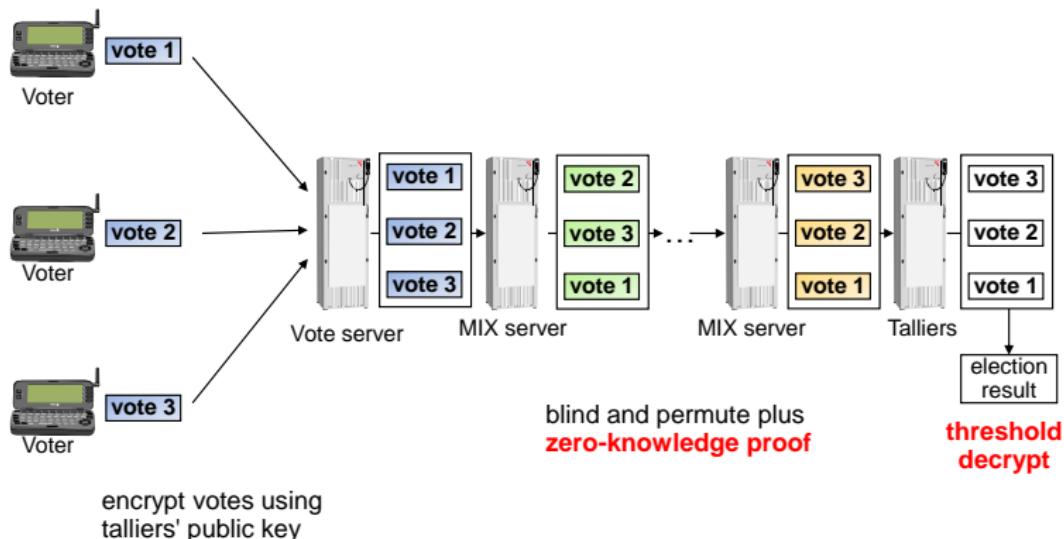
Cheating web server could

- pick an arbitrary function for π (e.g., with $\pi(1) = \pi(2)$)
- influence who picks who
- tell others who picked who
- ...

Q: How to do this by means of a cryptographic protocol—**no trusted party**?

A: Use a secure multiparty computation protocol!

Electronic Voting Using Verifiable Mixes



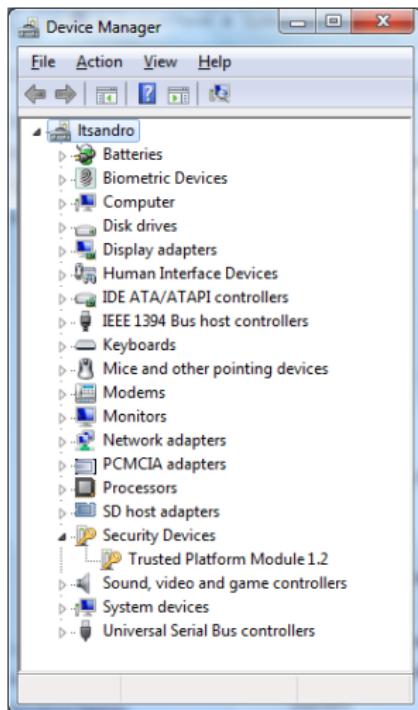
Zero-knowledge Proof in Your Laptop?

Direct Anonymous Attestation

Advanced Non-Interactive
Zero-Knowledge proof,
effectively a Group Signature

To prove that

- PC/laptop is authentic
(signing key belongs to
genuine “trusted computer”)
- but without giving away its
identity



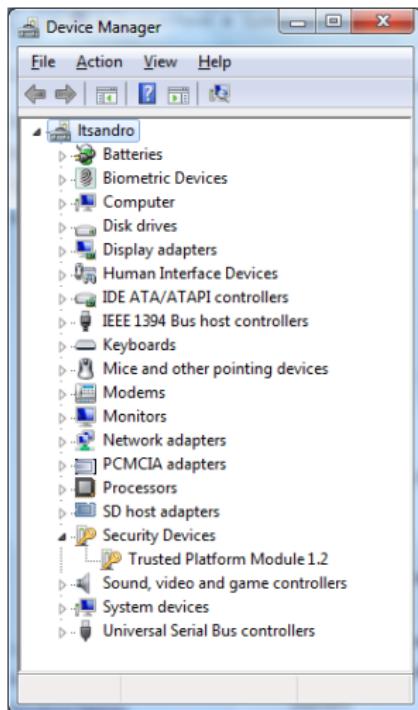
Zero-knowledge Proof in Your Laptop?

Direct Anonymous Attestation

Advanced **Non-Interactive
Zero-Knowledge proof**,
effectively a **Group Signature**

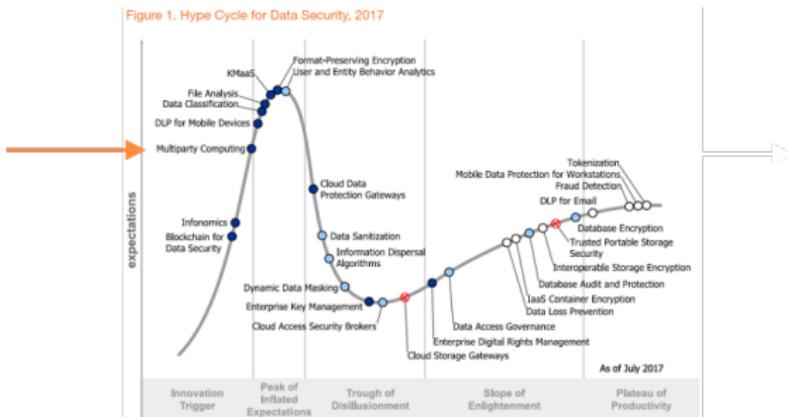
To prove that

- PC/laptop is **authentic** (signing key belongs to genuine “trusted computer”)
- but **without giving away its identity**



Secure Multiparty Computation Unleashed!

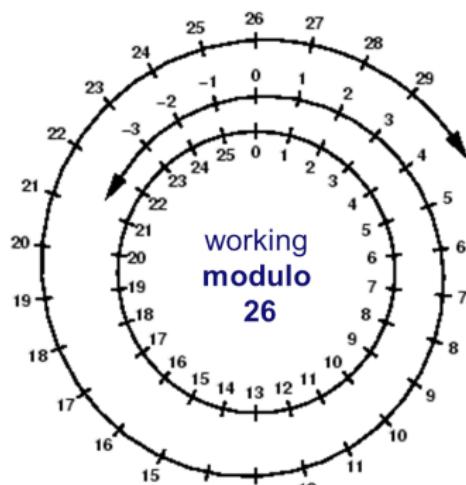
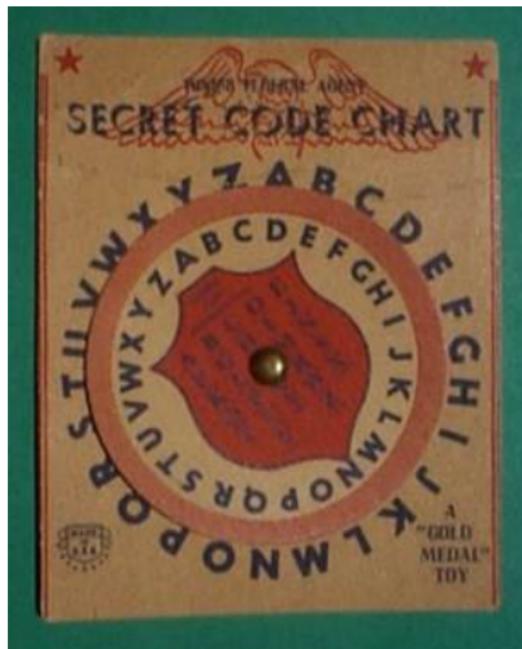
- In three 2017 Gartner Hype Cycle reports, see unboundtech.com.



- To implement blockchains like [Cardano](#).

Blockchains and cryptocurrencies utilize wealth of cryptographic protocols (implemented as open-source software, written in high-level programming languages, running on fast PCs/servers).

Crypto 1.0: Caesar Cipher



Rotate the alphabet by a fixed offset

Crypto 1.0

Crypto 1.0 concerns

- encryption and authentication of data,
- during communication and storage/retrieval

Crypto 1.0 primitives:

- Symmetric (secret key):
 - Stream/block ciphers
 - Message authentication codes
- Asymmetric (public key):
 - Public-key encryption
 - Digital signatures
 - Key-exchange protocols
- Keyless:
 - Cryptographic hash functions



Crypto 1.0

Crypto 1.0 concerns

- encryption and authentication of data,
- during communication and storage/retrieval

Crypto 1.0 primitives:

- Symmetric (secret key):
 - Stream/block ciphers
 - Message authentication codes
- Asymmetric (public key):
 - Public-key encryption
 - Digital signatures
 - Key-exchange protocols
- Keyless:
 - Cryptographic hash functions



Crypto 2.0

Crypto 2.0 additionally concerns

- computing with encrypted data,
- partial information release of data,
- hiding identity of data owners or any link with them.

Crypto 2.0 primitives:

- homomorphic encryption Rivest-Adleman-Dertouzos 1978, Gentry 2009
- secret sharing Blakley 1979, Shamir 1979
- blind signatures Chaum 1982
- oblivious transfer M. Rabin 1981, Even-Goldreich-Lempel 1985
- zero-knowledge proofs Goldwasser-Micali-Rackoff 1985, Goldreich-Micali-Wigderson 1986
- secure two/multi-party computation Yao 1982–1986, GMW 1987, BGW 1988, CCD 1988
- functional encryption Sahai-Waters 2005, Boneh-Sahai-Waters 2011, GKPZV 2013
- indistinguishability obfuscation Garg-Gentry-Halevi-Raykova-Sahai-Waters 2013, GMMSSZ 2016

Crypto 2.0

Crypto 2.0 additionally concerns

- computing with encrypted data,
- partial information release of data,
- hiding identity of data owners or any link with them.

Crypto 2.0 primitives:

- homomorphic encryption Rivest-Adleman-Dertouzos 1978, Gentry 2009
- secret sharing Blakley 1979, Shamir 1979
- blind signatures Chaum 1982
- oblivious transfer M. Rabin 1981, Even-Goldreich-Lempel 1985
- zero-knowledge proofs Goldwasser-Micali-Rackoff 1985, Goldreich-Micali-Wigderson 1986
- secure two/multi-party computation Yao 1982–1986, GMW 1987, BGW 1988, CCD 1988
- functional encryption Sahai-Waters 2005, Boneh-Sahai-Waters 2011, GKPZV 2013
- indistinguishability obfuscation Garg-Gentry-Halevi-Raykova-Sahai-Waters 2013, GMMSSZ 2016



Crypto 1.0 vs Crypto 2.0

Crypto 1.0 encryption and authentication:

- protect against **malicious outsiders**
 - attacks on storage or communication media
- can be achieved using symmetric crypto
- very high performance

Crypto 2.0 primitives additionally:

- protect against **malicious or corrupt insiders**
 - attacks by your protocol “partners”
- uses more powerful cryptographic primitives, essentially asymmetric crypto
- much harder to do efficiently



Crypto 1.0 vs Crypto 2.0

Crypto 1.0 encryption and authentication:

- protect against **malicious outsiders**
 - attacks on storage or communication media
- can be achieved using symmetric crypto
- very high performance

Crypto 2.0 primitives additionally:

- protect against **malicious or corrupt insiders**
 - attacks by your protocol “partners”
- uses more powerful cryptographic primitives, essentially asymmetric crypto
- much harder to do efficiently



Contents

1 Introduction

1.1 Terminology

1.2 Preliminaries

1.2.1 Number Theory

1.2.2 Group Theory

1.2.3 Probability Theory

1.2.4 Complexity Theory

1.3 Assumptions

1.3.1 Discrete Log and Diffie-Hellman Assumptions

1.3.2 Indistinguishability

1.3.3 Random Self-Reducibility

1.3.4 Random Oracle Model



Contents

2 Key Exchange Protocols

2.1 Diffie-Hellman Key Exchange

2.1.1 Basic Protocol

2.1.2 Passive Attacks

2.1.3 A Practical Variant

2.1.4 Aside: ElGamal Encryption

2.2 Authenticated Key Exchange

2.2.1 Man-in-the-Middle Attacks

2.2.2 A Protocol Using Digital Signatures

2.2.3 A Protocol Using Password-Based Encryption



Contents

3 Commitment Schemes

3.1 Definition

3.2 Examples

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

3.3 Homomorphic Commitments

Contents

4 Identification Protocols

4.1 Definitions

4.2 Password-based Schemes

4.3 One-Way Hash Chains

4.4 Basic Challenge-Response Protocols

4.4.1 Using Symmetric Encryption

4.4.2 Using Symmetric Authentication

4.4.3 Using Asymmetric Encryption

4.4.4 Using Asymmetric Authentication

4.5 Zero-knowledge Identification Protocols

4.5.1 Schnorr Zero-knowledge Protocol

4.5.2 Schnorr Protocol

4.5.3 Guillou-Quisquater Protocol

4.6 Witness Hiding Identification Protocols

4.6.1 Okamoto Protocol



Contents

5 Zero-Knowledge Proofs

5.1 Σ -Protocols

5.2 Composition of Σ -Protocols

5.2.1 Parallel Composition

5.2.2 AND-Composition

5.2.3 EQ-Composition

5.2.4 OR-Composition

5.2.5 NEQ-Composition

5.3 Miscellaneous Constructions

5.4 Non-interactive Σ -Proofs

5.4.1 Digital Signatures from Σ -Protocols

5.4.2 Proofs of Validity

5.4.3 Group Signatures



Contents

6 Threshold Cryptography

6.1 Secret Sharing

6.1.1 Shamir's Threshold Scheme

6.2 Verifiable Secret Sharing

6.2.1 Feldman VSS

6.2.2 Pedersen VSS

6.3 Threshold Cryptosystems

6.3.1 Threshold ElGamal Cryptosystem

Contents

7 Secure Multiparty Computation

7.1 Electronic Voting

7.2 Based on Threshold Homomorphic Cryptosystems

7.3 Based on Oblivious Transfer

Contents

8 Blind Signatures

8.1 Definition

8.2 Chaum Blind Signature Scheme

8.3 Blind Signatures from Σ -Protocols

Cryptology = Cryptography + Cryptanalysis

Field of **cryptology** is divided into two *mutually dependent* fields:

Cryptography: design of (mathematical) schemes related to information security which resist cryptanalysis.

Cryptanalysis: study of (mathematical) techniques for attacking cryptographic schemes.



IACR

International Association for **Cryptologic** Research

www.iacr.org

Cryptographic Algorithms/Protocols/Schemes

Definition 1.1

Cryptographic Algorithm: well-defined *transformation*, which on a given input value produces an output value, achieving certain security objectives.

Cryptographic Protocol: distributed algorithm describing precisely the *interactions between two or more entities*, achieving certain security objectives.

Cryptographic Scheme: suite of *related* cryptographic algorithms and cryptographic protocols, achieving certain security objectives.

Example

Digital signature schemes consist of three cryptographic algorithms:

- key generation
- signature generation
- signature verification



Cryptographic Algorithms/Protocols/Schemes

Definition 1.1

Cryptographic Algorithm: well-defined *transformation*, which on a given input value produces an output value, achieving certain security objectives.

Cryptographic Protocol: distributed algorithm describing precisely the *interactions between two or more entities*, achieving certain security objectives.

Cryptographic Scheme: suite of *related* cryptographic algorithms and cryptographic protocols, achieving certain security objectives.

Example

Digital signature schemes consist of three cryptographic algorithms:

- key generation
- signature generation
- signature verification



Cryptographic Algorithms/Protocols/Schemes

Definition 1.1

Cryptographic Algorithm: well-defined *transformation*, which on a given input value produces an output value, achieving certain security objectives.

Cryptographic Protocol: distributed algorithm describing precisely the *interactions between two or more entities*, achieving certain security objectives.

Cryptographic Scheme: suite of *related* cryptographic algorithms and cryptographic protocols, achieving certain security objectives.

Example

Digital signature schemes consist of three cryptographic algorithms:

- key generation
- signature generation
- signature verification



Cryptographic Algorithms/Protocols/Schemes

Definition 1.1

Cryptographic Algorithm: well-defined *transformation*, which on a given input value produces an output value, achieving certain security objectives.

Cryptographic Protocol: distributed algorithm describing precisely the *interactions between two or more entities*, achieving certain security objectives.

Cryptographic Scheme: suite of *related* cryptographic algorithms and cryptographic protocols, achieving certain security objectives.

Example

Digital signature schemes consist of three cryptographic algorithms:

- key generation
- signature generation
- signature verification



Communication Model

Entities in a cryptographic scheme are connected by **communication channels**.

Entities can be persons, organizations, devices, etc.

In cryptography, one often speaks of parties.

Entities interact in a cryptographic protocol by **exchanging messages** over *specific* communication channels.

Communication model describes mix of available channels:

- point-to-point channel
- broadcast channel
- private channel (secure channel)
- public channel (insecure channel)
- bulletin board



Communication Model

Entities in a cryptographic scheme are connected by **communication channels**.

Entities can be persons, organizations, devices, etc.

In cryptography, one often speaks of parties.

Entities interact in a cryptographic protocol by **exchanging messages** over *specific* communication channels.

Communication model describes mix of available channels:

- point-to-point channel
- broadcast channel
- private channel (secure channel)
- public channel (insecure channel)
- bulletin board



Communication Model

Entities in a cryptographic scheme are connected by **communication channels**.

Entities can be persons, organizations, devices, etc.

In cryptography, one often speaks of parties.

Entities interact in a cryptographic protocol by **exchanging messages** over *specific* communication channels.

Communication model describes mix of available channels:

- point-to-point channel
- broadcast channel
- private channel (secure channel)
- public channel (insecure channel)
- bulletin board

Adversaries and Attacks

Adversary = coalition of **attacker** and/or
corrupt entities of cryptographic scheme.

Attacker: **outsider**
Corrupt entities: **insiders.**

Definition 1.2

Passive Attack: adversary **does not interfere** with execution of algorithms/protocols. Passive adversary eavesdrops on communication between entities, and records all information, including all private information of corrupt entities.

Active attack: adversary **may—in addition—interfere** with communication within cryptographic scheme by deleting, injecting, or modifying messages; moreover, adversary **may have corrupt entities** deviate from prescribed behavior in arbitrary ways.

Adversaries and Attacks

Adversary = coalition of **attacker** and/or
corrupt entities of cryptographic scheme.

Attacker: **outsider**
Corrupt entities: **insiders.**

Definition 1.2

Passive Attack: adversary **does not interfere** with execution of algorithms/protocols. Passive adversary eavesdrops on communication between entities, and records all information, including all private information of corrupt entities.

Active attack: adversary **may—in addition—interfere** with communication within cryptographic scheme by deleting, injecting, or modifying messages; moreover, adversary **may have corrupt entities deviate** from prescribed behavior in arbitrary ways.

Attacks

Example (Passive Attacks)

- Eavesdropping on communication (by outsiders).
- Collecting data within a system (by insiders).
- Perform cryptanalysis on all acquired data.

Example (Active Attacks)

- Man-in-the-middle attacks ("grandmaster chess attack").
- Injection/deletion/modification of messages.
- Initiate/terminate protocol executions.
- Masquerading/spoofing attacks.
- Replay attacks.
- Reflection attacks.

Basic Number Theoretic Notions

For positive integer n :

\mathbb{Z}_n = “set of integers modulo n ”

$\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n : \gcd(x, n) = 1\}$ “integers with multiplicative inverse modulo n ”

$\phi(n) = |\mathbb{Z}_n^*| = n \prod_{p|n} (1 - 1/p)$ “Euler’s phi function”

Hence, $\phi(n)$ can be computed efficiently given the prime divisors of n .

Useful fact:

$$\frac{n}{\phi(n)} = \prod_{p|n} \frac{p}{p-1} = O(\log \log n)$$

Exercise 1.3

Prove the elementary bound $n/\phi(n) = O(\log n)$, using that the number of distinct prime factors $\omega(n)$ of n is $O(\log n)$, and that the i th smallest prime factor of n is at least $i + 1$ for $i = 1, 2, \dots, \omega(n)$.

Discrete Log Setting

Cyclic group of order n with **generator** g :

$$G_n = \langle g \rangle = \{1, g, g^2, g^3, \dots, g^{n-1}\}, \quad g^n = 1$$

x = log_g h: discrete log of $h \in G_n$ is unique $x \in \mathbb{Z}_n$ such that $h = g^x$.

Example 1.4 (Groups \mathbb{Z}_p^* , \mathbb{F}_q^*)

Integers modulo prime p : $G_n = \mathbb{Z}_p^*$ cyclic group of order $n = p - 1$.

Multiplicative group of order- q finite field: $G_n = \mathbb{F}_q^*$ cyclic, order $n = q - 1$.

Example 1.5 (Prime order subgroups of \mathbb{Z}_p^* , \mathbb{F}_q^*)

$G_n = \langle g \rangle$, with g an element of prime order p' in \mathbb{Z}_p^* , or in \mathbb{F}_q^* .

Then G_n is a cyclic group of prime order $n = p'$.

Example 1.6 (Group $E(\mathbb{F}_q)$ of \mathbb{F}_q -rational points on an elliptic curve E)

$E(\mathbb{F}_q)$ isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, hence not necessarily cyclic.

Take $G_n = E(\mathbb{F}_q)$ if $n_1 = 1$; otherwise, take a cyclic subgroup of $E(\mathbb{F}_q)$ for G_n .

Discrete Log Setting

Cyclic group of order n with **generator** g :

$$G_n = \langle g \rangle = \{1, g, g^2, g^3, \dots, g^{n-1}\}, \quad g^n = 1$$

x = log_g h: discrete log of $h \in G_n$ is unique $x \in \mathbb{Z}_n$ such that $h = g^x$.

Example 1.4 (Groups \mathbb{Z}_p^* , \mathbb{F}_q^*)

Integers modulo prime p : $G_n = \mathbb{Z}_p^*$ cyclic group of order $n = p - 1$.

Multiplicative group of order- q finite field: $G_n = \mathbb{F}_q^*$ cyclic, order $n = q - 1$.

Example 1.5 (Prime order subgroups of \mathbb{Z}_p^* , \mathbb{F}_q^*)

$G_n = \langle g \rangle$, with g an element of prime order p' in \mathbb{Z}_p^* , or in \mathbb{F}_q^* .

Then G_n is a cyclic group of prime order $n = p'$.

Example 1.6 (Group $E(\mathbb{F}_q)$ of \mathbb{F}_q -rational points on an elliptic curve E)

$E(\mathbb{F}_q)$ isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, hence not necessarily cyclic.

Take $G_n = E(\mathbb{F}_q)$ if $n_1 = 1$; otherwise, take a cyclic subgroup of $E(\mathbb{F}_q)$ for G_n .

Discrete Log Setting

Cyclic group of order n with **generator** g :

$$G_n = \langle g \rangle = \{1, g, g^2, g^3, \dots, g^{n-1}\}, \quad g^n = 1$$

x = log_g h: discrete log of $h \in G_n$ is unique $x \in \mathbb{Z}_n$ such that $h = g^x$.

Example 1.4 (Groups \mathbb{Z}_p^* , \mathbb{F}_q^*)

Integers modulo prime p : $G_n = \mathbb{Z}_p^*$ cyclic group of order $n = p - 1$.

Multiplicative group of order- q finite field: $G_n = \mathbb{F}_q^*$ cyclic, order $n = q - 1$.

Example 1.5 (Prime order subgroups of \mathbb{Z}_p^* , \mathbb{F}_q^*)

$G_n = \langle g \rangle$, with g an element of prime order p' in \mathbb{Z}_p^* , or in \mathbb{F}_q^* .

Then G_n is a cyclic group of prime order $n = p'$.

Example 1.6 (Group $E(\mathbb{F}_q)$ of \mathbb{F}_q -rational points on an elliptic curve E)

$E(\mathbb{F}_q)$ isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, hence not necessarily cyclic.

Take $G_n = E(\mathbb{F}_q)$ if $n_1 = 1$; otherwise, take a cyclic subgroup of $E(\mathbb{F}_q)$ for G_n .

Discrete Log Setting

Cyclic group of order n with **generator** g :

$$G_n = \langle g \rangle = \{1, g, g^2, g^3, \dots, g^{n-1}\}, \quad g^n = 1$$

x = log_g h: discrete log of $h \in G_n$ is unique $x \in \mathbb{Z}_n$ such that $h = g^x$.

Example 1.4 (Groups \mathbb{Z}_p^* , \mathbb{F}_q^*)

Integers modulo prime p : $G_n = \mathbb{Z}_p^*$ cyclic group of order $n = p - 1$.

Multiplicative group of order- q finite field: $G_n = \mathbb{F}_q^*$ cyclic, order $n = q - 1$.

Example 1.5 (Prime order subgroups of \mathbb{Z}_p^* , \mathbb{F}_q^*)

$G_n = \langle g \rangle$, with g an element of prime order p' in \mathbb{Z}_p^* , or in \mathbb{F}_q^* .

Then G_n is a cyclic group of prime order $n = p'$.

Example 1.6 (Group $E(\mathbb{F}_q)$ of \mathbb{F}_q -rational points on an elliptic curve E)

$E(\mathbb{F}_q)$ isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, hence not necessarily cyclic.

Take $G_n = E(\mathbb{F}_q)$ if $n_1 = 1$; otherwise, take a cyclic subgroup of $E(\mathbb{F}_q)$ for G_n .

Discrete Log Setting

Cyclic group of order n with **generator** g :

$$G_n = \langle g \rangle = \{1, g, g^2, g^3, \dots, g^{n-1}\}, \quad g^n = 1$$

x = log_g h: discrete log of $h \in G_n$ is unique $x \in \mathbb{Z}_n$ such that $h = g^x$.

Example 1.4 (Groups \mathbb{Z}_p^* , \mathbb{F}_q^*)

Integers modulo prime p : $G_n = \mathbb{Z}_p^*$ cyclic group of order $n = p - 1$.

Multiplicative group of order- q finite field: $G_n = \mathbb{F}_q^*$ cyclic, order $n = q - 1$.

Example 1.5 (Prime order subgroups of \mathbb{Z}_p^* , \mathbb{F}_q^*)

$G_n = \langle g \rangle$, with g an element of prime order p' in \mathbb{Z}_p^* , or in \mathbb{F}_q^* .

Then G_n is a cyclic group of prime order $n = p'$.

Example 1.6 (Group $E(\mathbb{F}_q)$ of \mathbb{F}_q -rational points on an elliptic curve E)

$E(\mathbb{F}_q)$ isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, hence not necessarily cyclic.

Take $G_n = E(\mathbb{F}_q)$ if $n_1 = 1$; otherwise, take a cyclic subgroup of $E(\mathbb{F}_q)$ for G_n .



Convenient Notation: $\langle g \rangle^*$

Definition (Set of all generators of $\langle g \rangle$)

$$\langle g \rangle^* = \{g^x : x \in \mathbb{Z}_n^*\}$$

Clearly: $|\langle g \rangle^*| = \phi(n)$.

Exercise 1.7

Show that for any $h \in \langle g \rangle$, the following conditions are equivalent:

- i) $h \in \langle g \rangle^*$,
- ii) $\text{ord}(h) = n$,
- iii) $\langle h \rangle = \langle g \rangle$,
- iv) $\langle h \rangle^* = \langle g \rangle^*$.

Exercise 1.8

Show that $a^{\log_h b} = b^{\log_h a}$ for any $a, b \in \langle g \rangle$ and $h \in \langle g \rangle^*$.

Notation

Discrete random variables X, Y, Z, \dots with a **finite** range V .

Probability distribution of X : (i) $0 \leq \Pr[X = v] \leq 1$ for all $v \in V$,
 (ii) $\sum_{v \in V} \Pr[X = v] = 1$,

$X \in_R V$: random variable X distributed **uniformly** on V , $\Pr[X = v] = 1/|V|$.

Example (Random bits)

$X \in_R \{0, 1\}$ denotes a uniformly random bit. Let $Y = 1 - X$.

- $X \neq Y$, in fact $\Pr[X = Y] = 0$,
- $\Pr[X = 0] = \Pr[Y = 0] = 1/2$, and
 $\Pr[X = 1] = \Pr[Y = 1] = 1/2$.

Example (Probability distributions)

- $\{u : u \in_R \{0, 1\}\} = \{0 \mapsto \frac{1}{2}, 1 \mapsto \frac{1}{2}\} = \{1 - u : u \in_R \{0, 1\}\}$
- $\{t + u : t, u \in_R \{0, 1\}\} = \{0 \mapsto \frac{1}{4}, 1 \mapsto \frac{1}{2}, 2 \mapsto \frac{1}{4}\}$
- $\{tu : t, u \in_R \{0, 1\}\} = \{0 \mapsto \frac{3}{4}, 1 \mapsto \frac{1}{4}\}$

Notation

Discrete random variables X, Y, Z, \dots with a **finite** range V .

Probability distribution of X : (i) $0 \leq \Pr[X = v] \leq 1$ for all $v \in V$,
 (ii) $\sum_{v \in V} \Pr[X = v] = 1$,

$X \in_R V$: random variable X distributed **uniformly** on V , $\Pr[X = v] = 1/|V|$.

Example (Random bits)

$X \in_R \{0, 1\}$ denotes a uniformly random bit. Let $Y = 1 - X$.

- $X \neq Y$, in fact $\Pr[X = Y] = 0$,
- $\Pr[X = 0] = \Pr[Y = 0] = 1/2$, and
 $\Pr[X = 1] = \Pr[Y = 1] = 1/2$.

Example (Probability distributions)

- $\{u : u \in_R \{0, 1\}\} = \{0 \mapsto \frac{1}{2}, 1 \mapsto \frac{1}{2}\} = \{1 - u : u \in_R \{0, 1\}\}$
- $\{t + u : t, u \in_R \{0, 1\}\} = \{0 \mapsto \frac{1}{4}, 1 \mapsto \frac{1}{2}, 2 \mapsto \frac{1}{4}\}$
- $\{tu : t, u \in_R \{0, 1\}\} = \{0 \mapsto \frac{3}{4}, 1 \mapsto \frac{1}{4}\}$

Notation

Discrete random variables X, Y, Z, \dots with a **finite** range V .

Probability distribution of X : (i) $0 \leq \Pr[X = v] \leq 1$ for all $v \in V$,
 (ii) $\sum_{v \in V} \Pr[X = v] = 1$,

$X \in_R V$: random variable X distributed **uniformly** on V , $\Pr[X = v] = 1/|V|$.

Example (Random bits)

$X \in_R \{0, 1\}$ denotes a uniformly random bit. Let $Y = 1 - X$.

- $X \neq Y$, in fact $\Pr[X = Y] = 0$,
- $\Pr[X = 0] = \Pr[Y = 0] = 1/2$, and
 $\Pr[X = 1] = \Pr[Y = 1] = 1/2$.

Example (Probability distributions)

- $\{u : u \in_R \{0, 1\}\} = \{0 \mapsto \frac{1}{2}, 1 \mapsto \frac{1}{2}\} = \{1 - u : u \in_R \{0, 1\}\}$
- $\{t + u : t, u \in_R \{0, 1\}\} = \{0 \mapsto \frac{1}{4}, 1 \mapsto \frac{1}{2}, 2 \mapsto \frac{1}{4}\}$
- $\{tu : t, u \in_R \{0, 1\}\} = \{0 \mapsto \frac{3}{4}, 1 \mapsto \frac{1}{4}\}$



Statistical Distance

Definition 1.9

The **statistical distance** $\Delta(X; Y)$ between random variables X and Y is defined as

$$\Delta(X; Y) = \frac{1}{2} \sum_{v \in V} |\Pr[X = v] - \Pr[Y = v]|,$$

where V denotes the set of possible values for X and Y .

Statistical distance is a bounded metric in the following sense.

Proposition 1.10

For random variables X, Y, Z :

- ➊ $0 \leq \Delta(X; Y) \leq 1$, “nonnegativity” and “boundedness”
- ➋ $\Delta(X; Y) = 0 \Leftrightarrow \forall_{v \in V} \Pr[X = v] = \Pr[Y = v]$, “identical distributions”
- ➌ $\Delta(X; Y) = \Delta(Y; X)$, “symmetry”
- ➍ $\Delta(X; Z) \leq \Delta(X; Y) + \Delta(Y; Z)$. “triangle inequality”



Statistical Distance

Definition 1.9

The **statistical distance** $\Delta(X; Y)$ between random variables X and Y is defined as

$$\Delta(X; Y) = \frac{1}{2} \sum_{v \in V} |\Pr[X = v] - \Pr[Y = v]|,$$

where V denotes the set of possible values for X and Y .

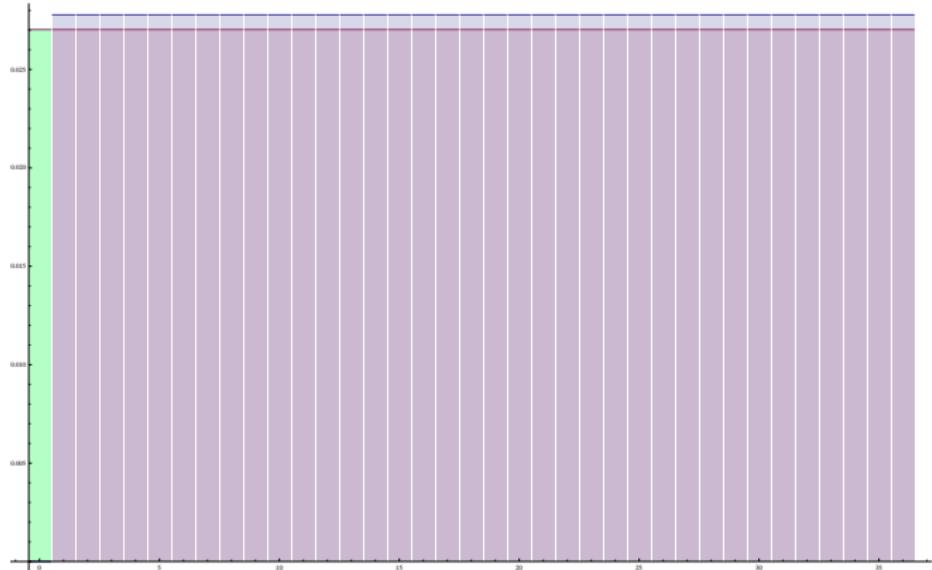
Statistical distance is a bounded metric in the following sense.

Proposition 1.10

For random variables X, Y, Z :

- ① $0 \leq \Delta(X; Y) \leq 1$, “nonnegativity” and “boundedness”
- ② $\Delta(X; Y) = 0 \Leftrightarrow \forall_{v \in V} \Pr[X = v] = \Pr[Y = v]$, “identical distributions”
- ③ $\Delta(X; Y) = \Delta(Y; X)$, “symmetry”
- ④ $\Delta(X; Z) \leq \Delta(X; Y) + \Delta(Y; Z)$. “triangle inequality”

House Edge (Advantage) in French Roulette



$$\Delta(X; Y) = \frac{1}{37} \approx 0.027 \text{ for } \begin{cases} X \in_R \{1, \dots, 36\} \\ Y \in_R \{0, \dots, 36\} \end{cases}$$

Statistical Distance: Equivalent Characterizations

Proposition 1.11

For random variables X and Y , $\Delta(X; Y)$ is equal to each of:

- (i) $\sum_{v \in V^+} \Pr[X = v] - \Pr[Y = v]$ with $V^+ = \{v \in V : \Pr[X = v] > \Pr[Y = v]\}$,
- (ii) $\sum_{v \in V} \Pr[X = v] \doteq \Pr[Y = v]$ with $x \doteq y = \max(x - y, 0)$,
- (iii) $1 - \sum_{v \in V} \min(\Pr[X = v], \Pr[Y = v])$,
- (iv) $\max_{W \subseteq V} |\Pr[X \in W] - \Pr[Y \in W]|$.

Exercise 1.12

(a) Prove Proposition 1.10. (b) Prove Proposition 1.11.

Exercise 1.13

Prove that $\Delta(f(X); f(Y)) \leq \Delta(X; Y)$ for any function f defined on V .

Statistical Distance: Equivalent Characterizations

Proposition 1.11

For random variables X and Y , $\Delta(X; Y)$ is equal to each of:

- ① $\sum_{v \in V^+} \Pr[X = v] - \Pr[Y = v]$ with $V^+ = \{v \in V : \Pr[X = v] > \Pr[Y = v]\}$,
- ② $\sum_{v \in V} \Pr[X = v] \dot{-} \Pr[Y = v]$ with $x \dot{-} y = \max(x - y, 0)$,
- ③ $1 - \sum_{v \in V} \min(\Pr[X = v], \Pr[Y = v])$,
- ④ $\max_{W \subseteq V} |\Pr[X \in W] - \Pr[Y \in W]|$.

Exercise 1.12

(a) Prove Proposition 1.10. (b) Prove Proposition 1.11.

Exercise 1.13

Prove that $\Delta(f(X); f(Y)) \leq \Delta(X; Y)$ for any function f defined on V .

Statistical Distance: Equivalent Characterizations

Proposition 1.11

For random variables X and Y , $\Delta(X; Y)$ is equal to each of:

- ④ $\sum_{v \in V^+} \Pr[X = v] - \Pr[Y = v]$ with $V^+ = \{v \in V : \Pr[X = v] > \Pr[Y = v]\}$,
- ⑤ $\sum_{v \in V} \Pr[X = v] \dot{-} \Pr[Y = v]$ with $x \dot{-} y = \max(x - y, 0)$,
- ⑥ $1 - \sum_{v \in V} \min(\Pr[X = v], \Pr[Y = v])$,
- ⑦ $\max_{W \subseteq V} |\Pr[X \in W] - \Pr[Y \in W]|$.

Exercise 1.12

(a) Prove Proposition 1.10. (b) Prove Proposition 1.11.

Exercise 1.13

Prove that $\Delta(f(X); f(Y)) \leq \Delta(X; Y)$ for any function f defined on V .

Statistical Distance: Equivalent Characterizations

Proposition 1.11

For random variables X and Y , $\Delta(X; Y)$ is equal to each of:

- ④ $\sum_{v \in V^+} \Pr[X = v] - \Pr[Y = v]$ with $V^+ = \{v \in V : \Pr[X = v] > \Pr[Y = v]\}$,
- ⑤ $\sum_{v \in V} \Pr[X = v] \dot{-} \Pr[Y = v]$ with $x \dot{-} y = \max(x - y, 0)$,
- ⑥ $1 - \sum_{v \in V} \min(\Pr[X = v], \Pr[Y = v])$,
- ⑦ $\max_{W \subseteq V} |\Pr[X \in W] - \Pr[Y \in W]|$.

Exercise 1.12

(a) Prove Proposition 1.10. (b) Prove Proposition 1.11.

Exercise 1.13

Prove that $\Delta(f(X); f(Y)) \leq \Delta(X; Y)$ for any function f defined on V .

Statistical Distance: Equivalent Characterizations

Proposition 1.11

For random variables X and Y , $\Delta(X; Y)$ is equal to each of:

- ① $\sum_{v \in V^+} \Pr[X = v] - \Pr[Y = v]$ with $V^+ = \{v \in V : \Pr[X = v] > \Pr[Y = v]\}$,
- ② $\sum_{v \in V} \Pr[X = v] \dot{-} \Pr[Y = v]$ with $x \dot{-} y = \max(x - y, 0)$,
- ③ $1 - \sum_{v \in V} \min(\Pr[X = v], \Pr[Y = v])$,
- ④ $\max_{W \subseteq V} |\Pr[X \in W] - \Pr[Y \in W]|$.

Exercise 1.12

(a) Prove Proposition 1.10. (b) Prove Proposition 1.11.

Exercise 1.13

Prove that $\Delta(f(X); f(Y)) \leq \Delta(X; Y)$ for any function f defined on V .

Statistical Distance: Equivalent Characterizations

Proposition 1.11

For random variables X and Y , $\Delta(X; Y)$ is equal to each of:

- ① $\sum_{v \in V^+} \Pr[X = v] - \Pr[Y = v]$ with $V^+ = \{v \in V : \Pr[X = v] > \Pr[Y = v]\}$,
- ② $\sum_{v \in V} \Pr[X = v] \dot{-} \Pr[Y = v]$ with $x \dot{-} y = \max(x - y, 0)$,
- ③ $1 - \sum_{v \in V} \min(\Pr[X = v], \Pr[Y = v])$,
- ④ $\max_{W \subseteq V} |\Pr[X \in W] - \Pr[Y \in W]|$.

Exercise 1.12

(a) Prove Proposition 1.10. (b) Prove Proposition 1.11.

Exercise 1.13

Prove that $\Delta(f(X); f(Y)) \leq \Delta(X; Y)$ for any function f defined on V .

Statistical Distance: More Exercises

Exercise 1.14

For $n, d \geq 1$, consider distributions X and Y given by

$$\begin{aligned} X &= \{u : u \in_R \{0, \dots, n-1\}, \\ Y &= \{u + d : u \in_R \{0, \dots, n-1\}\}. \end{aligned}$$

Compute $\Delta(X; Y)$, assuming $d \leq n$. Also, what is $\Delta(X; Y)$ if $d > n$?

Exercise 1.15

For $n \geq 1$, consider distributions X, Y, Z given by

$$\begin{aligned} X &= \{u : u \in_R \{0, \dots, n-1\}\} \\ Y &= \{2u : u \in_R \{0, \dots, n-1\}\} \\ Z &= \{2u + 1 : u \in_R \{0, \dots, n-1\}\}. \end{aligned}$$

Show that $\Delta(Y; Z) = 1$. Show that $\Delta(X; Y) = \Delta(X; Z) = 1/2$ for even n , and also determine $\Delta(X; Y)$ and $\Delta(X; Z)$ for odd n .



Statistical Distance: More Exercises

Exercise 1.16

For $n \geq 1$, let $X \in_R \mathbb{Z}_n$ and $Y \in_R \mathbb{Z}_n^*$. (a) Determine $\Delta(X; Y)$. (b) Show that $\Delta(X + Y; XY) = 0$, where addition and multiplication are done modulo n .

Exercise 1.17

For n prime, let h and M_0 be arbitrary, fixed elements of $G_n = \langle g \rangle$, $h \neq 1$. Consider distributions X , Y , and Z given by

$$\begin{aligned} X &= \{(A, B) : A \in_R \langle g \rangle, B \in_R \langle g \rangle\}, \\ Y &= \{(g^u, h^u M) : u \in_R \mathbb{Z}_n, m \in_R \langle g \rangle\}, \\ Z &= \{(g^u, h^u M_0) : u \in_R \mathbb{Z}_n\}. \end{aligned}$$

Show that $\Delta(X; Y) = 0$ and that $\Delta(Y; Z) = 1 - 1/n$. Show also that $\Delta(X; Z) = 1 - 1/n$, using triangle inequalities.

Exercise 1.18

For $n \geq d \geq 1$, let random variable X take on values in $\{0, \dots, d-1\}$, and let $U \in_R \{0, \dots, n-1\}$. Show $\Delta(U; X+U) \leq (d-1)/n$, and that this bound is tight.



Complexity Classes P vs NP

P: deterministic **Polynomial** time

“easy to solve”

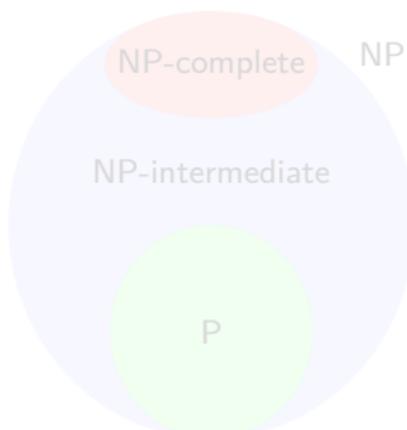
NP: **Non-deterministic Polynomial** time

“easy to check”

Clearly: $P \subseteq NP$

Long-standing **conjecture**: $P \neq NP$

\$1,000,000 Millennium Prize Problem
 of the Clay Mathematics Institute



Problems in P: easy, e.g., Sorting, Primality, ...

NP-intermediate problems: DL, Factoring, Graph Isomorphism (all conjectured!)

NP-complete problems: hardest in NP, e.g., 3SAT, Hamiltonian Circuit, ...

(NP-hard problems: \geq NP-complete, e.g., Traveling Salesman, ...)

Complexity Classes P vs NP

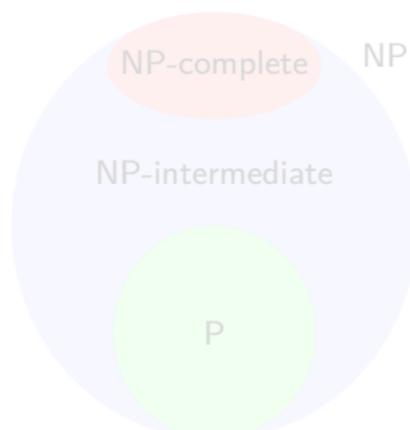
P: deterministic **Polynomial** time

“easy to solve”

NP: **Non-deterministic Polynomial** time

“easy to check”

Clearly: $P \subseteq NP$



Long-standing **conjecture**: $P \neq NP$

\$1,000,000 Millennium Prize Problem
 of the Clay Mathematics Institute

Problems in P: easy, e.g., Sorting, Primality, ...

NP-intermediate problems: DL, Factoring, Graph Isomorphism (all conjectured!)

NP-complete problems: hardest in NP, e.g., 3SAT, Hamiltonian Circuit, ...

(NP-hard problems: \geq NP-complete, e.g., Traveling Salesman, ...)

Complexity Classes P vs NP

P: deterministic **Polynomial** time

“easy to solve”

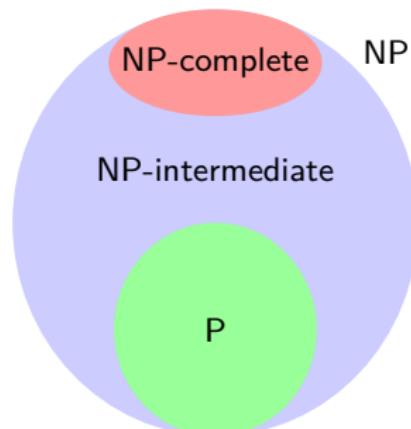
NP: **Non-deterministic Polynomial** time

“easy to check”

Clearly: $P \subseteq NP$

Long-standing **conjecture**: $P \neq NP$

\$1,000,000 Millennium Prize Problem
of the Clay Mathematics Institute



Problems in P: easy, e.g., Sorting, Primality, ...

NP-intermediate problems: DL, Factoring, Graph Isomorphism (all conjectured!)

NP-complete problems: hardest in NP, e.g., 3SAT, Hamiltonian Circuit, ...

(NP-hard problems: \geq NP-complete, e.g., Traveling Salesman, ...)

Complexity Classes P vs NP

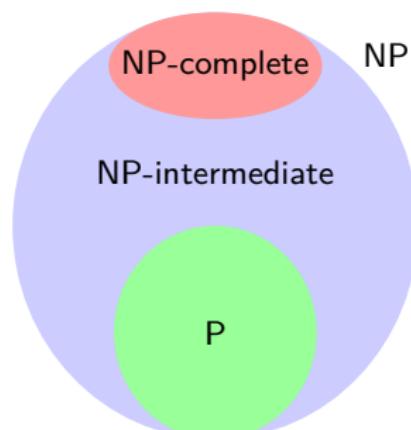
P: deterministic **Polynomial** time

“easy to solve”

NP: **Non-deterministic Polynomial** time

“easy to check”

Clearly: $P \subseteq NP$



Long-standing **conjecture**: $P \neq NP$

\$1,000,000 Millennium Prize Problem
of the Clay Mathematics Institute

Problems in P: easy, e.g., Sorting, Primality, ...

NP-intermediate problems: DL, Factoring, Graph Isomorphism (all conjectured!)

NP-complete problems: hardest in NP, e.g., 3SAT, Hamiltonian Circuit, ...

(NP-hard problems: \geq NP-complete, e.g., Traveling Salesman, ...)

Worst-Case vs Average-Case Complexity

NP-complete problems: hard in the worst case.

Over **all** problem instances of a given size.

But **potentially** easy in the average case!!

Uniformly random problem instances of a given size may be easy.
(Chances of hitting worst-case may be negligible.)

Worst-Case vs Average-Case Complexity

NP-complete problems: hard in the worst case.

Over **all** problem instances of a given size.

But **potentially** easy in the average case!!

Uniformly random problem instances of a given size may be easy.
(Chances of hitting worst-case may be negligible.)

Probabilistic Turing Machines

Church's thesis: “Turing machines are powerful enough to describe any kind of algorithm”

Turing machine: finite control part plus an infinite data tape.

Three important types of Turing machines:

- Deterministic: unique successor configuration.
- Non-deterministic: one of several possible successor configurations.
- Probabilistic: one of several possible successor configurations, chosen uniformly at random.

Probabilistic Turing machines \Leftrightarrow probabilistic/randomized algorithms
(deterministic Turing machines extended with random tape)

Probabilistic Turing Machines

Church's thesis: “Turing machines are powerful enough to describe any kind of algorithm”

Turing machine: finite control part plus an infinite data tape.

Three important types of Turing machines:

- **Deterministic:** unique successor configuration.
- **Non-deterministic:** one of several possible successor configurations.
- **Probabilistic:** one of several possible successor configurations,
chosen uniformly at random.

Probabilistic Turing machines \Leftrightarrow probabilistic/randomized algorithms
(deterministic Turing machines extended with random tape)

Probabilistic Turing Machines

Church's thesis: “Turing machines are powerful enough to describe any kind of algorithm”

Turing machine: finite control part plus an infinite data tape.

Three important types of Turing machines:

- **Deterministic:** unique successor configuration.
- **Non-deterministic:** one of several possible successor configurations.
- **Probabilistic:** one of several possible successor configurations,
chosen uniformly at random.

Probabilistic Turing machines \Leftrightarrow probabilistic/randomized algorithms
(deterministic Turing machines extended with random tape)

Complexity Class BPP

Complexity class BPP: problems solved with **Bounded-error**
by **Probabilistic Polynomial time (p.p.t.)** Turing machines

For every YES-instance: $\Pr[\text{"accept"}] \geq 2/3$

For every NO-instance: $\Pr[\text{"accept"}] \leq 1/3$

Probability over all internal random choices (random tapes).

Clearly: $P \subseteq BPP$

Conjectured: $P = BPP$

Thus, BPP class of “easy” (efficiently solvable) problems.

Note: relation of BPP versus NP not known.



Complexity Class BPP

Complexity class BPP: problems solved with **Bounded-error**
by **Probabilistic Polynomial time (p.p.t.)** Turing machines

For every YES-instance: $\Pr[\text{"accept"}] \geq 2/3$

For every NO-instance: $\Pr[\text{"accept"}] \leq 1/3$

Probability over all internal random choices (random tapes).

Clearly: $P \subseteq BPP$

Conjectured: $P = BPP$

Thus, BPP class of “easy” (efficiently solvable) problems.

Note: relation of BPP versus NP not known.



Discrete Log and Diffie-Hellman Assumptions

Definition 1.19 (Discrete Logarithm (DL) assumption)

For group $\langle g \rangle$, hard to compute x given random group element g^x .

Definition 1.20 ((Computational) Diffie-Hellman (DH) assumption)

For group $\langle g \rangle$, hard to compute g^{xy} given random group elements g^x, g^y .

Definition 1.21 (Decisional Diffie-Hellman (DDH) assumption)

For group $\langle g \rangle$, hard to distinguish g^{xy} from random group element g^z given random group elements g^x, g^y .

Evidently: DDH assumption \Rightarrow DH assumption \Rightarrow DL assumption.

Discrete Log and Diffie-Hellman Assumptions

Definition 1.19 (Discrete Logarithm (DL) assumption)

For group $\langle g \rangle$, hard to compute x given random group element g^x .

Definition 1.20 ((Computational) Diffie-Hellman (DH) assumption)

For group $\langle g \rangle$, hard to compute g^{xy} given random group elements g^x, g^y .

Definition 1.21 (Decisional Diffie-Hellman (DDH) assumption)

For group $\langle g \rangle$, hard to distinguish g^{xy} from random group element g^z given random group elements g^x, g^y .

Evidently: DDH assumption \Rightarrow DH assumption \Rightarrow DL assumption.

Discrete Log and Diffie-Hellman Assumptions

Definition 1.19 (Discrete Logarithm (DL) assumption)

For group $\langle g \rangle$, hard to compute x given random group element g^x .

Definition 1.20 ((Computational) Diffie-Hellman (DH) assumption)

For group $\langle g \rangle$, hard to compute g^{xy} given random group elements g^x, g^y .

Definition 1.21 (Decisional Diffie-Hellman (DDH) assumption)

For group $\langle g \rangle$, hard to distinguish g^{xy} from random group element g^z given random group elements g^x, g^y .

Evidently: DDH assumption \Rightarrow DH assumption \Rightarrow DL assumption.

Discrete Log and Diffie-Hellman Assumptions

Definition 1.19 (Discrete Logarithm (DL) assumption)

For group $\langle g \rangle$, hard to compute x given random group element g^x .

Definition 1.20 ((Computational) Diffie-Hellman (DH) assumption)

For group $\langle g \rangle$, hard to compute g^{xy} given random group elements g^x, g^y .

Definition 1.21 (Decisional Diffie-Hellman (DDH) assumption)

For group $\langle g \rangle$, hard to distinguish g^{xy} from random group element g^z given random group elements g^x, g^y .

Evidently: DDH assumption \Rightarrow DH assumption \Rightarrow DL assumption.

Negligible Function

Definition 1.22

Nonnegative function $f : \mathbb{N} \rightarrow \mathbb{R}$ is **negligible** if for every $\gamma \in \mathbb{N}$ there exists a $k_0 \in \mathbb{N}$ such that for all $k \geq k_0$, $f(k) \leq 1/k^\gamma$.

Example

Negligible:

- $1/2^k = 2^{-k}$ “exponentially fast to 0”
- $1/2^{\sqrt{k}} = 2^{-\sqrt{k}}$ “sub-exponentially fast to 0”
- $1/k^{\log k} = k^{-\log k}$ “quasi-polynomially fast to 0”

Not negligible:

- $1/k = k^{-1}$ “linearly fast to 0”
- $1/k^2 = k^{-2}$ “quadratically fast to 0”
- $1/k^{100} = k^{-100}$ “polynomially fast to 0”



Indistinguishability

Definition 1.23

Let $X = \{X_i\}$ and $Y = \{Y_i\}$ be two families of random variables.

Then X and Y are:

- ① perfectly indistinguishable if $\Delta(X_i; Y_i) = 0$
- ② statistically indistinguishable if $\Delta(X_i; Y_i)$ is negligible
- ③ computationally indistinguishable if $\Delta(D(X_i); D(Y_i))$ is negligible for all p.p.t. Boolean distinguishers D

Advantage $\text{Adv}_D(X_i, Y_i) = |\Pr[D(X_i) = 1] - \Pr[D(Y_i) = 1]|$

Exercise 1.24

Show that $\text{Adv}_D(X_i, Y_i) = \Delta(D(X_i); D(Y_i))$ for any Boolean distinguisher D .

Exercise 1.25

Show that computational indistinguishability is implied by statistical indistinguishability. Hint: use Proposition 1.11(iv), cf. Exercise 1.13.

Indistinguishability

Definition 1.23

Let $X = \{X_i\}$ and $Y = \{Y_i\}$ be two families of random variables.

Then X and Y are:

- ④ **perfectly indistinguishable** if $\Delta(X_i; Y_i) = 0$
- ⑤ **statistically indistinguishable** if $\Delta(X_i; Y_i)$ is negligible
- ⑥ **computationally indistinguishable** if $\Delta(D(X_i); D(Y_i))$ is negligible for all p.p.t. Boolean distinguishers D

Advantage $\text{Adv}_D(X_i, Y_i) = |\Pr[D(X_i) = 1] - \Pr[D(Y_i) = 1]|$

Exercise 1.24

Show that $\text{Adv}_D(X_i, Y_i) = \Delta(D(X_i); D(Y_i))$ for any Boolean distinguisher D .

Exercise 1.25

Show that computational indistinguishability is implied by statistical indistinguishability. Hint: use Proposition 1.11(iv), cf. Exercise 1.13.



Indistinguishability

Definition 1.23

Let $X = \{X_i\}$ and $Y = \{Y_i\}$ be two families of random variables.

Then X and Y are:

- ④ **perfectly indistinguishable** if $\Delta(X_i; Y_i) = 0$
- ⑤ **statistically indistinguishable** if $\Delta(X_i; Y_i)$ is negligible
- ⑥ **computationally indistinguishable** if $\Delta(D(X_i); D(Y_i))$ is negligible for all p.p.t. Boolean distinguishers D

Advantage $\text{Adv}_D(X_i, Y_i) = |\Pr[D(X_i) = 1] - \Pr[D(Y_i) = 1]|$

Exercise 1.24

Show that $\text{Adv}_D(X_i, Y_i) = \Delta(D(X_i); D(Y_i))$ for any Boolean distinguisher D .

Exercise 1.25

Show that computational indistinguishability is implied by statistical indistinguishability. Hint: use Proposition 1.11(iv), cf. Exercise 1.13.

Indistinguishability

Definition 1.23

Let $X = \{X_i\}$ and $Y = \{Y_i\}$ be two families of random variables.

Then X and Y are:

- ④ **perfectly indistinguishable** if $\Delta(X_i; Y_i) = 0$
- ⑤ **statistically indistinguishable** if $\Delta(X_i; Y_i)$ is negligible
- ⑥ **computationally indistinguishable** if $\Delta(D(X_i); D(Y_i))$ is negligible for all p.p.t. Boolean distinguishers D

Advantage $\text{Adv}_D(X_i, Y_i) = |\Pr[D(X_i) = 1] - \Pr[D(Y_i) = 1]|$

Exercise 1.24

Show that $\text{Adv}_D(X_i, Y_i) = \Delta(D(X_i); D(Y_i))$ for any Boolean distinguisher D .

Exercise 1.25

Show that computational indistinguishability is implied by statistical indistinguishability. Hint: use Proposition 1.11(iv), cf. Exercise 1.13.



Indistinguishability

Definition 1.23

Let $X = \{X_i\}$ and $Y = \{Y_i\}$ be two families of random variables.

Then X and Y are:

- ④ **perfectly indistinguishable** if $\Delta(X_i; Y_i) = 0$
- ⑤ **statistically indistinguishable** if $\Delta(X_i; Y_i)$ is negligible
- ⑥ **computationally indistinguishable** if $\Delta(D(X_i); D(Y_i))$ is negligible for all p.p.t. Boolean distinguishers D

Advantage $\text{Adv}_D(X_i, Y_i) = |\Pr[D(X_i) = 1] - \Pr[D(Y_i) = 1]|$

Exercise 1.24

Show that $\text{Adv}_D(X_i, Y_i) = \Delta(D(X_i); D(Y_i))$ for any Boolean distinguisher D .

Exercise 1.25

Show that computational indistinguishability is implied by statistical indistinguishability. Hint: use Proposition 1.11(iv), cf. Exercise 1.13.



Indistinguishability

Definition 1.23

Let $X = \{X_i\}$ and $Y = \{Y_i\}$ be two families of random variables.

Then X and Y are:

- ④ **perfectly indistinguishable** if $\Delta(X_i; Y_i) = 0$
- ⑤ **statistically indistinguishable** if $\Delta(X_i; Y_i)$ is negligible
- ⑥ **computationally indistinguishable** if $\Delta(D(X_i); D(Y_i))$ is negligible for all p.p.t. Boolean distinguishers D

Advantage $\text{Adv}_D(X_i, Y_i) = |\Pr[D(X_i) = 1] - \Pr[D(Y_i) = 1]|$

Exercise 1.24

Show that $\text{Adv}_D(X_i, Y_i) = \Delta(D(X_i); D(Y_i))$ for any Boolean distinguisher D .

Exercise 1.25

Show that computational indistinguishability is implied by statistical indistinguishability. Hint: use Proposition 1.11(iv), cf. Exercise 1.13.



Boolean Distinguisher D



Distinguisher samples from fixed (but unknown) “source”, either X_i or Y_i .

For computational indistinguishability, distinguisher should fail using any p.p.t. **statistical test**.

DDH Assumption (Definition 1.21)

$$\begin{aligned} X_{\langle g \rangle} &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} && \text{"DH triples"} \\ Y_{\langle g \rangle} &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\} && \text{"non-DH triples"} \end{aligned}$$

DDH assumption: $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ are computationally indistinguishable.
 Also possible to take:

$$Y'_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n\} \quad \text{"random triples"}$$

$$\begin{aligned} \Delta(Y; Y') &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z = xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} ((n^3 - n^2) \left| \frac{1}{n^3 - n^2} - \frac{1}{n^3} \right| + n^2 \left| 0 - \frac{1}{n^3} \right|) \\ &= 1/n. \end{aligned}$$

Since n is exponentially large in security parameter,
 statistical distance of $1/n$ is negligible.

DDH Assumption (Definition 1.21)

$$\begin{aligned} X_{\langle g \rangle} &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} && \text{"DH triples"} \\ Y_{\langle g \rangle} &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\} && \text{"non-DH triples"} \end{aligned}$$

DDH assumption: $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ are computationally indistinguishable.

Also possible to take:

$$Y'_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n\} \quad \text{"random triples"}$$

$$\begin{aligned} \Delta(Y; Y') &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z=xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} ((n^3 - n^2) \left| \frac{1}{n^3 - n^2} - \frac{1}{n^3} \right| + n^2 \left| 0 - \frac{1}{n^3} \right|) \\ &= 1/n. \end{aligned}$$

Since n is exponentially large in security parameter,
statistical distance of $1/n$ is negligible.

DDH Assumption (Definition 1.21)

$$\begin{aligned} X_{\langle g \rangle} &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} && \text{"DH triples"} \\ Y_{\langle g \rangle} &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\} && \text{"non-DH triples"} \end{aligned}$$

DDH assumption: $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ are computationally indistinguishable.

Also possible to take:

$$Y'_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n\} \quad \text{"random triples"}$$

$$\begin{aligned} \Delta(Y; Y') &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z = xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} ((n^3 - n^2) \left| \frac{1}{n^3 - n^2} - \frac{1}{n^3} \right| + n^2 \left| 0 - \frac{1}{n^3} \right|) \\ &= 1/n. \end{aligned}$$

Since n is exponentially large in security parameter,
statistical distance of $1/n$ is negligible.

DDH Assumption (Definition 1.21)

$$\begin{aligned} X_{\langle g \rangle} &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} && \text{"DH triples"} \\ Y_{\langle g \rangle} &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\} && \text{"non-DH triples"} \end{aligned}$$

DDH assumption: $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ are computationally indistinguishable.
 Also possible to take:

$$Y'_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n\} \quad \text{"random triples"}$$

$$\begin{aligned} \Delta(Y; Y') &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z=xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left((n^3 - n^2) \left| \frac{1}{n^3 - n^2} - \frac{1}{n^3} \right| + n^2 \left| 0 - \frac{1}{n^3} \right| \right) \\ &= 1/n. \end{aligned}$$

Since n is exponentially large in security parameter,
 statistical distance of $1/n$ is negligible.

DDH Assumption (Definition 1.21)

$$\begin{aligned} X_{\langle g \rangle} &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} && \text{"DH triples"} \\ Y_{\langle g \rangle} &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\} && \text{"non-DH triples"} \end{aligned}$$

DDH assumption: $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ are computationally indistinguishable.
 Also possible to take:

$$Y'_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n\} \quad \text{"random triples"}$$

$$\begin{aligned} \Delta(Y; Y') &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z = xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} ((n^3 - n^2) \left| \frac{1}{n^3 - n^2} - \frac{1}{n^3} \right| + n^2 \left| 0 - \frac{1}{n^3} \right|) \\ &= 1/n. \end{aligned}$$

Since n is exponentially large in security parameter,
 statistical distance of $1/n$ is negligible.

DDH Assumption (Definition 1.21)

$$\begin{aligned} X_{\langle g \rangle} &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} && \text{"DH triples"} \\ Y_{\langle g \rangle} &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\} && \text{"non-DH triples"} \end{aligned}$$

DDH assumption: $X_{\langle g \rangle}$ and $Y_{\langle g \rangle}$ are computationally indistinguishable.
 Also possible to take:

$$Y'_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n\} \quad \text{"random triples"}$$

$$\begin{aligned} \Delta(Y; Y') &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z = xy} \left| \Pr[Y = (g^x, g^y, g^z)] - \Pr[Y' = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} ((n^3 - n^2) \left| \frac{1}{n^3 - n^2} - \frac{1}{n^3} \right| + n^2 \left| 0 - \frac{1}{n^3} \right|) \\ &= 1/n. \end{aligned}$$

Since n is exponentially large in security parameter,
 statistical distance of $1/n$ is negligible.

DDH Assumption

$$\begin{aligned} X &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} \\ Y &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\}. \end{aligned}$$

$$\begin{aligned} \Delta(X; Y) &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z=xy} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(n^2 \left| \frac{1}{n^2} - 0 \right| + (n^3 - n^2) \left| 0 - \frac{1}{n^3 - n^2} \right| \right) \\ &= 1. \end{aligned}$$

$\Delta(X; Y) = 1$ not surprising, since X and Y are disjoint!

Triangle inequality: $\Delta(X; Y') \geq \Delta(X; Y) - \Delta(Y; Y') = 1 - 1/n$.

Exercise 1.26

Check that actually $\Delta(X; Y') = 1 - 1/n$.

DDH Assumption

$$\begin{aligned} X &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} \\ Y &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\}. \end{aligned}$$

$$\begin{aligned} \Delta(X; Y) &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z=xy} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(n^2 \left| \frac{1}{n^2} - 0 \right| + (n^3 - n^2) \left| 0 - \frac{1}{n^3 - n^2} \right| \right) \\ &= 1. \end{aligned}$$

$\Delta(X; Y) = 1$ not surprising, since X and Y are disjoint!

Triangle inequality: $\Delta(X; Y') \geq \Delta(X; Y) - \Delta(Y; Y') = 1 - 1/n$.

Exercise 1.26

Check that actually $\Delta(X; Y') = 1 - 1/n$.



DDH Assumption

$$\begin{aligned} X &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} \\ Y &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\}. \end{aligned}$$

$$\begin{aligned} \Delta(X; Y) &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z=xy} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} (n^2 \left| \frac{1}{n^2} - 0 \right| + (n^3 - n^2) \left| 0 - \frac{1}{n^3 - n^2} \right|) \\ &= 1. \end{aligned}$$

$\Delta(X; Y) = 1$ not surprising, since X and Y are disjoint!

Triangle inequality: $\Delta(X; Y') \geq \Delta(X; Y) - \Delta(Y; Y') = 1 - 1/n$.

Exercise 1.26

Check that actually $\Delta(X; Y') = 1 - 1/n$.



DDH Assumption

$$\begin{aligned} X &= \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\} \\ Y &= \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\}. \end{aligned}$$

$$\begin{aligned} \Delta(X; Y) &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} \left(\sum_{x,y,z \in \mathbb{Z}_n, z=xy} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right. \\ &\quad \left. + \sum_{x,y,z \in \mathbb{Z}_n, z \neq xy} \left| \Pr[X = (g^x, g^y, g^z)] - \Pr[Y = (g^x, g^y, g^z)] \right| \right) \\ &= \frac{1}{2} (n^2 \left| \frac{1}{n^2} - 0 \right| + (n^3 - n^2) \left| 0 - \frac{1}{n^3 - n^2} \right|) \\ &= 1. \end{aligned}$$

$\Delta(X; Y) = 1$ not surprising, since X and Y are disjoint!

Triangle inequality: $\Delta(X; Y') \geq \Delta(X; Y) - \Delta(Y; Y') = 1 - 1/n$.

Exercise 1.26

Check that actually $\Delta(X; Y') = 1 - 1/n$.



Random Self-Reducibility

Definition 1.27

Problem is (**perfectly**) random self-reducible if any instance I can be solved by these three steps:

- ① Transform instance I into uniformly random instance I' .
- ② Solve instance I' .
- ③ Extract solution for I from solution for I' .

Only steps 1 and 3 are required to run in polynomial time.

Commonly random self-reducible:

- DL/DH/DDH problems for *fixed* DL setting $\langle g \rangle$.
- RSA-based problems for *fixed* RSA modulus m .

Likely to be **not** random self-reducible:

- Factoring integers of fixed bit length.
- Any NP-complete problem.



Random Self-Reducibility

Definition 1.27

Problem is (**perfectly**) **random self-reducible** if any instance I can be solved by these three steps:

- ① Transform instance I into *uniformly random* instance I' .
- ② Solve instance I' .
- ③ Extract solution for I from solution for I' .

Only steps 1 and 3 are required to run in polynomial time.

Commonly random self-reducible:

- DL/DH/DDH problems for *fixed* DL setting $\langle g \rangle$.
- RSA-based problems for *fixed* RSA modulus m .

Likely to be **not** random self-reducible:

- Factoring integers of fixed bit length.
- Any NP-complete problem.



Random Self-Reducibility

Definition 1.27

Problem is (**perfectly**) **random self-reducible** if any instance I can be solved by these three steps:

- ① Transform instance I into *uniformly random* instance I' .
- ② Solve instance I' .
- ③ Extract solution for I from solution for I' .

Only steps 1 and 3 are required to run in polynomial time.

Commonly random self-reducible:

- DL/DH/DDH problems for *fixed* DL setting $\langle g \rangle$.
- RSA-based problems for *fixed* RSA modulus m .

Likely to be **not** random self-reducible:

- Factoring integers of fixed bit length.
- Any NP-complete problem.



Random Self-Reducibility

Definition 1.27

Problem is (**perfectly**) **random self-reducible** if any instance I can be solved by these three steps:

- ① Transform instance I into *uniformly random* instance I' .
- ② Solve instance I' .
- ③ Extract solution for I from solution for I' .

Only steps 1 and 3 are required to run in polynomial time.

Commonly random self-reducible:

- DL/DH/DDH problems for *fixed* DL setting $\langle g \rangle$.
- RSA-based problems for *fixed* RSA modulus m .

Likely to be **not** random self-reducible:

- Factoring integers of fixed bit length.
- Any NP-complete problem.



Random Self-Reducibility

Definition 1.27

Problem is (**perfectly**) **random self-reducible** if any instance I can be solved by these three steps:

- ① Transform instance I into *uniformly random* instance I' .
- ② Solve instance I' .
- ③ Extract solution for I from solution for I' .

Only steps 1 and 3 are required to run in polynomial time.

Commonly random self-reducible:

- DL/DH/DDH problems for *fixed* DL setting $\langle g \rangle$.
- RSA-based problems for *fixed* RSA modulus m .

Likely to be **not** random self-reducible:

- Factoring integers of fixed bit length.
- Any NP-complete problem.



Random Self-Reducibility

Definition 1.27

Problem is (**perfectly**) **random self-reducible** if any instance I can be solved by these three steps:

- ① Transform instance I into *uniformly random* instance I' .
- ② Solve instance I' .
- ③ Extract solution for I from solution for I' .

Only steps 1 and 3 are required to run in polynomial time.

Commonly random self-reducible:

- DL/DH/DDH problems for *fixed* DL setting $\langle g \rangle$.
- RSA-based problems for *fixed* RSA modulus m .

Likely to be **not** random self-reducible:

- Factoring integers of fixed bit length.
- Any NP-complete problem.

Random Self-Reducibility

Definition 1.27

Problem is (**perfectly**) **random self-reducible** if any instance I can be solved by these three steps:

- ① Transform instance I into *uniformly random* instance I' .
- ② Solve instance I' .
- ③ Extract solution for I from solution for I' .

Only steps 1 and 3 are required to run in polynomial time.

Commonly random self-reducible:

- DL/DH/DDH problems for *fixed* DL setting $\langle g \rangle$.
- RSA-based problems for *fixed* RSA modulus m .

Likely to be **not** random self-reducible:

- Factoring integers of fixed bit length.
- Any NP-complete problem.



Reduction to Average Case

Proposition 1.28 (Informal)

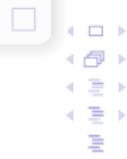
Any random self-reducible problem that is hard in the worst-case is also hard on the average.

Proof (sketch).

Suppose random self-reducible problem is easy on the average.

Then any problem instance can be solved easily:

- ① Transform given instance into a uniformly random instance.
- ② Solve it (easy because it's the average case).
- ③ Extract solution to original problem.



Note that the algorithm in the proof itself is probabilistic.

Reduction to Average Case

Proposition 1.28 (Informal)

Any random self-reducible problem that is hard in the worst-case is also hard on the average.

Proof (sketch).

Suppose random self-reducible problem is easy on the average.

Then any problem instance can be solved easily:

- ① Transform given instance into a uniformly random instance.
- ② Solve it (easy because it's the average case).
- ③ Extract solution to original problem.

Note that the algorithm in the proof itself is probabilistic.



Reduction to Average Case

Proposition 1.28 (Informal)

Any random self-reducible problem that is hard in the worst-case is also hard on the average.

Proof (sketch).

Suppose random self-reducible problem is easy on the average.

Then any problem instance can be solved easily:

- ① Transform given instance into a uniformly random instance.
- ② Solve it (easy because it's the average case).
- ③ Extract solution to original problem.

Note that the algorithm in the proof itself is probabilistic.



Reduction to Average Case

Proposition 1.28 (Informal)

Any random self-reducible problem that is hard in the worst-case is also hard on the average.

Proof (sketch).

Suppose random self-reducible problem is easy on the average.

Then any problem instance can be solved easily:

- ① Transform given instance into a uniformly random instance.
- ② Solve it (easy because it's the average case).
- ③ Extract solution to original problem.

Note that the algorithm in the proof itself is probabilistic.



Reduction to Average Case

Proposition 1.28 (Informal)

Any random self-reducible problem that is hard in the worst-case is also hard on the average.

Proof (sketch).

Suppose random self-reducible problem is easy on the average.

Then any problem instance can be solved easily:

- ① Transform given instance into a uniformly random instance.
- ② Solve it (easy because it's the average case).
- ③ Extract solution to original problem.

Note that the algorithm in the proof itself is probabilistic.



Reduction to Average Case

Proposition 1.28 (Informal)

Any random self-reducible problem that is hard in the worst-case is also hard on the average.

Proof (sketch).

Suppose random self-reducible problem is easy on the average.

Then any problem instance can be solved easily:

- ① Transform given instance into a uniformly random instance.
- ② Solve it (easy because it's the average case).
- ③ Extract solution to original problem.



Note that the algorithm in the proof itself is probabilistic.

Reduction to Average Case

Proposition 1.28 (Informal)

Any random self-reducible problem that is hard in the worst-case is also hard on the average.

Proof (sketch).

Suppose random self-reducible problem is easy on the average.

Then any problem instance can be solved easily:

- ① Transform given instance into a uniformly random instance.
- ② Solve it (easy because it's the average case).
- ③ Extract solution to original problem.



Note that the algorithm in the proof itself is probabilistic.

Random Self-Reducible Discrete Log Problems

We consider a fixed **group $\langle g \rangle$ of any order $n \geq 1$** .

Definition (DL, DH, DDH problems)

DL Compute x , given g^x with $x \in \mathbb{Z}_n$.

DH Compute g^{xy} , given g^x, g^y with $x, y \in \mathbb{Z}_n$.

DDH Distinguish g^{xy} from g^z , given g^x, g^y with $x, y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

For **n prime**, DDH problem corresponds to distinguishing (disjoint) distributions

$$X_{\langle g \rangle} = \{(g^x, g^y, g^{xy}) : x, y \in_R \mathbb{Z}_n\}$$

$$Y_{\langle g \rangle} = \{(g^x, g^y, g^z) : x, y, z \in_R \mathbb{Z}_n, z \neq xy\}$$

Proposition 1.29

DL, DH, and DDH problems are random self-reducible.



Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ① Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ① Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ② Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ③ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$.

Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ① Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ① Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ② Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ③ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$.



Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ① Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ① Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ② Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ③ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$. □

Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ① Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ① Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ② Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ③ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$. □



Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ① Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ① Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ② Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ③ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$. □

Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ➊ Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ➋ Solve instance h' yielding $x' = \log_g h'$.
- ➌ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ➊ Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ➋ Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ➌ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$.



Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ① Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ① Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ② Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ③ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$. □

Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ➊ Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ➋ Solve instance h' yielding $x' = \log_g h'$.
- ➌ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ➊ Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ➋ Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ➌ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$.



Proof (DL, DH problems are random self-reducible).

Given any DL problem instance $h = g^x$ with $x \in \mathbb{Z}_n$:

- ① Transform h into uniformly random instance $h' = hg^u$ with $u \in_R \mathbb{Z}_n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Given any DH problem instance $I = (g^x, g^y)$ with $x, y \in \mathbb{Z}_n$:

- ① Transform I into $I' = (g^{x'}, g^{y'}) = (g^x g^t, g^y g^u)$ with $t, u \in_R \mathbb{Z}_n$.
- ② Solve instance I' yielding $g^{x'y'} = g^{(x+t)(y+u)} = g^{xy+xu+ty+tu}$.
- ③ Extract solution as $g^{xy} = g^{x'y'} / ((g^x)^u (g^y)^t g^{tu})$.

Note: $I' = (g^{x'}, g^{y'})$ is distributed uniformly on $\langle g \rangle \times \langle g \rangle$. □

Proof (DDH problem is random self-reducible).

Given any DDH problem instance $I = (g^x, g^y, g^z)$ with $x, y, z \in \mathbb{Z}_n$:

- ① Transform I into $I' = (g^{xs+t}, g^{y+u}, g^{zs+xsu+yt+tu})$, with $s \in_R \mathbb{Z}_n^*$, $t, u \in_R \mathbb{Z}_n$.
- ② Solve instance I' yielding bit b' .
- ③ Output $b = b'$.

Since $s \in \mathbb{Z}_n^*$, $I' = (g^{x'}, g^{y'}, g^{z'})$ satisfies $z' = x'y'$ iff I satisfies $z = xy$:

$$z' - x'y' = zs + xsu + yt + tu - (xs + t)(y + u) = (z - xy)s.$$

If $z = xy$, triple I' is uniform among all triples $(g^{x'}, g^{y'}, g^{x'y'})$: for $v, w \in \mathbb{Z}_n$

$$\Pr[x' = v, y' = w, z' = vw] = \frac{1}{n^2}.$$

If $z - xy \in \mathbb{Z}_n^*$, s, t, u determined by $s = (z' - x'y')/(z - xy)$, $t = x' - xs$, $u = y' - y$, hence I' uniform among all $(g^{x'}, g^{y'}, g^{z'})$ with $z' - x'y' \in \mathbb{Z}_n^*$:

$$\Pr[x' = v, y' = w, z' = r] = \Pr[s = \frac{r - vw}{z - xy}, t = v - xs, u = w - y] = \frac{1}{\phi(n)n^2},$$

for any $r, v, w \in \mathbb{Z}_n$ with $r - vw \in \mathbb{Z}_n^*$:



Proof (DDH problem is random self-reducible).

Given any DDH problem instance $I = (g^x, g^y, g^z)$ with $x, y, z \in \mathbb{Z}_n$:

- ❶ Transform I into $I' = (g^{xs+t}, g^{y+u}, g^{zs+xsu+yt+tu})$, with $s \in_R \mathbb{Z}_n^*$, $t, u \in_R \mathbb{Z}_n$.
- ❷ Solve instance I' yielding bit b' .
- ❸ Output $b = b'$.

Since $s \in \mathbb{Z}_n^*$, $I' = (g^{x'}, g^{y'}, g^{z'})$ satisfies $z' = x'y'$ iff I satisfies $z = xy$:

$$z' - x'y' = zs + xsu + yt + tu - (xs + t)(y + u) = (z - xy)s.$$

If $z = xy$, triple I' is uniform among all triples $(g^{x'}, g^{y'}, g^{x'y'})$: for $v, w \in \mathbb{Z}_n$

$$\Pr[x' = v, y' = w, z' = vw] = \frac{1}{n^2}.$$

If $z - xy \in \mathbb{Z}_n^*$, s, t, u determined by $s = (z' - x'y')/(z - xy)$, $t = x' - xs$, $u = y' - y$, hence I' uniform among all $(g^{x'}, g^{y'}, g^{z'})$ with $z' - x'y' \in \mathbb{Z}_n^*$:

$$\Pr[x' = v, y' = w, z' = r] = \Pr[s = \frac{r - vw}{z - xy}, t = v - xs, u = w - y] = \frac{1}{\phi(n)n^2},$$

for any $r, v, w \in \mathbb{Z}_n$ with $r - vw \in \mathbb{Z}_n^*$:



Proof (DDH problem is random self-reducible).

Given any DDH problem instance $I = (g^x, g^y, g^z)$ with $x, y, z \in \mathbb{Z}_n$:

- ❶ Transform I into $I' = (g^{xs+t}, g^{y+u}, g^{zs+xsu+yt+tu})$, with $s \in_R \mathbb{Z}_n^*$, $t, u \in_R \mathbb{Z}_n$.
- ❷ Solve instance I' yielding bit b' .
- ❸ Output $b = b'$.

Since $s \in \mathbb{Z}_n^*$, $I' = (g^{x'}, g^{y'}, g^{z'})$ satisfies $z' = x'y'$ iff I satisfies $z = xy$:

$$z' - x'y' = zs + xsu + yt + tu - (xs + t)(y + u) = (z - xy)s.$$

If $z = xy$, triple I' is uniform among all triples $(g^{x'}, g^{y'}, g^{x'y'})$: for $v, w \in \mathbb{Z}_n$

$$\Pr[x' = v, y' = w, z' = vw] = \frac{1}{n^2}.$$

If $z - xy \in \mathbb{Z}_n^*$, s, t, u determined by $s = (z' - x'y')/(z - xy)$, $t = x' - xs$, $u = y' - y$, hence I' uniform among all $(g^{x'}, g^{y'}, g^{z'})$ with $z' - x'y' \in \mathbb{Z}_n^*$:

$$\Pr[x' = v, y' = w, z' = r] = \Pr[s = \frac{r - vw}{z - xy}, t = v - xs, u = w - y] = \frac{1}{\phi(n)n^2},$$

for any $r, v, w \in \mathbb{Z}_n$ with $r - vw \in \mathbb{Z}_n^*$:



Proof (DDH problem is random self-reducible).

Given any DDH problem instance $I = (g^x, g^y, g^z)$ with $x, y, z \in \mathbb{Z}_n$:

- ❶ Transform I into $I' = (g^{xs+t}, g^{y+u}, g^{zs+xsu+yt+tu})$, with $s \in_R \mathbb{Z}_n^*$, $t, u \in_R \mathbb{Z}_n$.
- ❷ Solve instance I' yielding bit b' .
- ❸ Output $b = b'$.

Since $s \in \mathbb{Z}_n^*$, $I' = (g^{x'}, g^{y'}, g^{z'})$ satisfies $z' = x'y'$ iff I satisfies $z = xy$:

$$z' - x'y' = zs + xsu + yt + tu - (xs + t)(y + u) = (z - xy)s.$$

If $z = xy$, triple I' is uniform among all triples $(g^{x'}, g^{y'}, g^{x'y'})$: for $v, w \in \mathbb{Z}_n$

$$\Pr[x' = v, y' = w, z' = vw] = \frac{1}{n^2}.$$

If $z - xy \in \mathbb{Z}_n^*$, s, t, u determined by $s = (z' - x'y')/(z - xy)$, $t = x' - xs$, $u = y' - y$, hence I' uniform among all $(g^{x'}, g^{y'}, g^{z'})$ with $z' - x'y' \in \mathbb{Z}_n^*$:

$$\Pr[x' = v, y' = w, z' = r] = \Pr[s = \frac{r - vw}{z - xy}, t = v - xs, u = w - y] = \frac{1}{\phi(n)n^2},$$

for any $r, v, w \in \mathbb{Z}_n$ with $r - vw \in \mathbb{Z}_n^*$:



Proof (DDH problem is random self-reducible).

Given any DDH problem instance $I = (g^x, g^y, g^z)$ with $x, y, z \in \mathbb{Z}_n$:

- ❶ Transform I into $I' = (g^{xs+t}, g^{y+u}, g^{zs+xsu+yt+tu})$, with $s \in_R \mathbb{Z}_n^*$, $t, u \in_R \mathbb{Z}_n$.
- ❷ Solve instance I' yielding bit b' .
- ❸ Output $b = b'$.

Since $s \in \mathbb{Z}_n^*$, $I' = (g^{x'}, g^{y'}, g^{z'})$ satisfies $z' = x'y'$ iff I satisfies $z = xy$:

$$z' - x'y' = zs + xsu + yt + tu - (xs + t)(y + u) = (z - xy)s.$$

If $z = xy$, triple I' is uniform among all triples $(g^{x'}, g^{y'}, g^{x'y'})$: for $v, w \in \mathbb{Z}_n$

$$\Pr[x' = v, y' = w, z' = vw] = \frac{1}{n^2}.$$

If $z - xy \in \mathbb{Z}_n^*$, s, t, u determined by $s = (z' - x'y')/(z - xy)$, $t = x' - xs$, $u = y' - y$, hence I' uniform among all $(g^{x'}, g^{y'}, g^{z'})$ with $z' - x'y' \in \mathbb{Z}_n^*$:

$$\Pr[x' = v, y' = w, z' = r] = \Pr[s = \frac{r - vw}{z - xy}, t = v - xs, u = w - y] = \frac{1}{\phi(n)n^2},$$

for any $r, v, w \in \mathbb{Z}_n$ with $r - vw \in \mathbb{Z}_n^*$:



Proof (DDH problem is random self-reducible).

Given any DDH problem instance $I = (g^x, g^y, g^z)$ with $x, y, z \in \mathbb{Z}_n$:

- ❶ Transform I into $I' = (g^{xs+t}, g^{y+u}, g^{zs+xsu+yt+tu})$, with $s \in_R \mathbb{Z}_n^*$, $t, u \in_R \mathbb{Z}_n$.
- ❷ Solve instance I' yielding bit b' .
- ❸ Output $b = b'$.

Since $s \in \mathbb{Z}_n^*$, $I' = (g^{x'}, g^{y'}, g^{z'})$ satisfies $z' = x'y'$ iff I satisfies $z = xy$:

$$z' - x'y' = zs + xsu + yt + tu - (xs + t)(y + u) = (z - xy)s.$$

If $z = xy$, triple I' is uniform among all triples $(g^{x'}, g^{y'}, g^{x'y'})$: for $v, w \in \mathbb{Z}_n$

$$\Pr[x' = v, y' = w, z' = vw] = \frac{1}{n^2}.$$

If $z - xy \in \mathbb{Z}_n^*$, s, t, u determined by $s = (z' - x'y')/(z - xy)$, $t = x' - xs$, $u = y' - y$, hence I' uniform among all $(g^{x'}, g^{y'}, g^{z'})$ with $z' - x'y' \in \mathbb{Z}_n^*$:

$$\Pr[x' = v, y' = w, z' = r] = \Pr[s = \frac{r - vw}{z - xy}, t = v - xs, u = w - y] = \frac{1}{\phi(n)n^2},$$

for any $r, v, w \in \mathbb{Z}_n$ with $r - vw \in \mathbb{Z}_n^*$:



Proof (DDH problem is random self-reducible).

Given any DDH problem instance $I = (g^x, g^y, g^z)$ with $x, y, z \in \mathbb{Z}_n$:

- ➊ Transform I into $I' = (g^{xs+t}, g^{y+u}, g^{zs+xsu+yt+tu})$, with $s \in_R \mathbb{Z}_n^*$, $t, u \in_R \mathbb{Z}_n$.
- ➋ Solve instance I' yielding bit b' .
- ➌ Output $b = b'$.

Since $s \in \mathbb{Z}_n^*$, $I' = (g^{x'}, g^{y'}, g^{z'})$ satisfies $z' = x'y'$ iff I satisfies $z = xy$:

$$z' - x'y' = zs + xsu + yt + tu - (xs + t)(y + u) = (z - xy)s.$$

If $z = xy$, triple I' is uniform among all triples $(g^{x'}, g^{y'}, g^{x'y'})$: for $v, w \in \mathbb{Z}_n$

$$\Pr[x' = v, y' = w, z' = vw] = \frac{1}{n^2}.$$

If $z - xy \in \mathbb{Z}_n^*$, s, t, u determined by $s = (z' - x'y')/(z - xy)$, $t = x' - xs$, $u = y' - y$, hence I' uniform among all $(g^{x'}, g^{y'}, g^{z'})$ with $z' - x'y' \in \mathbb{Z}_n^*$:

$$\Pr[x' = v, y' = w, z' = r] = \Pr[s = \frac{r - vw}{z - xy}, t = v - xs, u = w - y] = \frac{1}{\phi(n)n^2},$$

for any $r, v, w \in \mathbb{Z}_n$ with $r - vw \in \mathbb{Z}_n^*$:



Random Self-Reducible DL Variants

Definition (DL*, DH*, DH, DDH*, DDH** problems)**

DL* Compute x , given g^x with $x \in \mathbb{Z}_n^*$.

DH* Compute g^{xy} , given g^x, g^y with $x \in \mathbb{Z}_n^*$ and $y \in \mathbb{Z}_n$.

DH** Compute g^{xy} , given g^x, g^y with $x, y \in \mathbb{Z}_n^*$.

DDH* Distinguish g^{xy} from g^z , given g^x, g^y with $x \in \mathbb{Z}_n^*$, $y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

DDH** Distinguish g^{xy} from g^z , given g^x, g^y with $x, y, z \in \mathbb{Z}_n^*$, $z - xy \in \mathbb{Z}_n^*$.

Example 1.30 (DL* problem is random self-reducible)

Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

- ➊ Transform h into a *uniformly random* instance $h' = h^u$ with $u \in_R \mathbb{Z}_n^*$.
- ➋ Solve instance h' yielding $x' = \log_g h'$.
- ➌ Extract solution as $x = \log_g h = x'/u \bmod n$.

Note: h' is distributed uniformly on $\langle g \rangle^*$.



Random Self-Reducible DL Variants

Definition (DL*, DH*, DH, DDH*, DDH** problems)**

DL* Compute x , given g^x with $x \in \mathbb{Z}_n^*$.

DH* Compute g^{xy} , given g^x, g^y with $x \in \mathbb{Z}_n^*$ and $y \in \mathbb{Z}_n$.

DH** Compute g^{xy} , given g^x, g^y with $x, y \in \mathbb{Z}_n^*$.

DDH* Distinguish g^{xy} from g^z , given g^x, g^y with $x \in \mathbb{Z}_n^*$, $y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

DDH** Distinguish g^{xy} from g^z , given g^x, g^y with $x, y, z \in \mathbb{Z}_n^*$, $z - xy \in \mathbb{Z}_n^*$.

Example 1.30 (DL* problem is random self-reducible)

Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

- ① Transform h into a *uniformly random* instance $h' = h^u$ with $u \in_R \mathbb{Z}_n^*$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x'/u \bmod n$.

Note: h' is distributed uniformly on $\langle g \rangle^*$.



Random Self-Reducible DL Variants

Definition (DL*, DH*, DH, DDH*, DDH** problems)**

DL* Compute x , given g^x with $x \in \mathbb{Z}_n^*$.

DH* Compute g^{xy} , given g^x, g^y with $x \in \mathbb{Z}_n^*$ and $y \in \mathbb{Z}_n$.

DH** Compute g^{xy} , given g^x, g^y with $x, y \in \mathbb{Z}_n^*$.

DDH* Distinguish g^{xy} from g^z , given g^x, g^y with $x \in \mathbb{Z}_n^*$, $y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

DDH** Distinguish g^{xy} from g^z , given g^x, g^y with $x, y, z \in \mathbb{Z}_n^*$, $z - xy \in \mathbb{Z}_n^*$.

Example 1.30 (DL* problem is random self-reducible)

Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

- ➊ Transform h into a *uniformly random* instance $h' = h^u$ with $u \in_R \mathbb{Z}_n^*$.
- ➋ Solve instance h' yielding $x' = \log_g h'$.
- ➌ Extract solution as $x = \log_g h = x'/u \bmod n$.

Note: h' is distributed uniformly on $\langle g \rangle^*$.



Random Self-Reducible DL Variants

Definition (DL*, DH*, DH, DDH*, DDH** problems)**

DL* Compute x , given g^x with $x \in \mathbb{Z}_n^*$.

DH* Compute g^{xy} , given g^x, g^y with $x \in \mathbb{Z}_n^*$ and $y \in \mathbb{Z}_n$.

DH** Compute g^{xy} , given g^x, g^y with $x, y \in \mathbb{Z}_n^*$.

DDH* Distinguish g^{xy} from g^z , given g^x, g^y with $x \in \mathbb{Z}_n^*$, $y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

DDH** Distinguish g^{xy} from g^z , given g^x, g^y with $x, y, z \in \mathbb{Z}_n^*$, $z - xy \in \mathbb{Z}_n^*$.

Example 1.30 (DL* problem is random self-reducible)

Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

- ❶ Transform h into a *uniformly random* instance $h' = h^u$ with $u \in_R \mathbb{Z}_n^*$.
- ❷ Solve instance h' yielding $x' = \log_g h'$.
- ❸ Extract solution as $x = \log_g h = x'/u \bmod n$.

Note: h' is distributed uniformly on $\langle g \rangle^*$.



Random Self-Reducible DL Variants

Definition (DL*, DH*, DH, DDH*, DDH** problems)**

DL* Compute x , given g^x with $x \in \mathbb{Z}_n^*$.

DH* Compute g^{xy} , given g^x, g^y with $x \in \mathbb{Z}_n^*$ and $y \in \mathbb{Z}_n$.

DH** Compute g^{xy} , given g^x, g^y with $x, y \in \mathbb{Z}_n^*$.

DDH* Distinguish g^{xy} from g^z , given g^x, g^y with $x \in \mathbb{Z}_n^*$, $y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

DDH** Distinguish g^{xy} from g^z , given g^x, g^y with $x, y, z \in \mathbb{Z}_n^*$, $z - xy \in \mathbb{Z}_n^*$.

Example 1.30 (DL* problem is random self-reducible)

Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

- ❶ Transform h into a *uniformly random* instance $h' = h^u$ with $u \in_R \mathbb{Z}_n^*$.
- ❷ Solve instance h' yielding $x' = \log_g h'$.
- ❸ Extract solution as $x = \log_g h = x'/u \bmod n$.

Note: h' is distributed uniformly on $\langle g \rangle^*$.



Random Self-Reducible DL Variants

Definition (DL*, DH*, DH, DDH*, DDH** problems)**

DL* Compute x , given g^x with $x \in \mathbb{Z}_n^*$.

DH* Compute g^{xy} , given g^x, g^y with $x \in \mathbb{Z}_n^*$ and $y \in \mathbb{Z}_n$.

DH** Compute g^{xy} , given g^x, g^y with $x, y \in \mathbb{Z}_n^*$.

DDH* Distinguish g^{xy} from g^z , given g^x, g^y with $x \in \mathbb{Z}_n^*$, $y, z \in \mathbb{Z}_n$, $z - xy \in \mathbb{Z}_n^*$.

DDH** Distinguish g^{xy} from g^z , given g^x, g^y with $x, y, z \in \mathbb{Z}_n^*$, $z - xy \in \mathbb{Z}_n^*$.

Example 1.30 (DL* problem is random self-reducible)

Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

- ❶ Transform h into a *uniformly random* instance $h' = h^u$ with $u \in_R \mathbb{Z}_n^*$.
- ❷ Solve instance h' yielding $x' = \log_g h'$.
- ❸ Extract solution as $x = \log_g h = x'/u \bmod n$.

Note: h' is distributed uniformly on $\langle g \rangle^*$.

Example (cont.)

Alternatively, use transformation for DL problem in proof of Proposition 1.28.

Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

- ① Transform h into uniformly random $h' = hg^u$, $u \in_R \mathbb{Z}_n$ until $\text{ord}(h') = n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Condition $\text{ord}(h') = n$ ensures $h' \in \langle g \rangle^*$ in step 2, cf. Exercise 1.7.

In fact, h' is distributed uniformly on $\langle g \rangle^*$ as required.

Test $\text{ord}(h') = n$ can be evaluated efficiently, given prime factorization of n .

Expected running time of step 1 polynomial in size of n : on average

$n/\phi(n) = O(\log \log n)$ uniformly random $h' \in \langle g \rangle$ needed to find one $h' \in \langle g \rangle^*$.

Note: if n is prime, $\text{ord}(h') = n$ is equivalent to $h' \neq 1$.



Example (cont.)

Alternatively, use transformation for DL problem in proof of Proposition 1.28.

Given any instance $h = g^x$ with $x \in \mathbb{Z}_n^*$:

- ① Transform h into uniformly random $h' = hg^u$, $u \in_R \mathbb{Z}_n$ until $\text{ord}(h') = n$.
- ② Solve instance h' yielding $x' = \log_g h'$.
- ③ Extract solution as $x = \log_g h = x' - u \bmod n$.

Condition $\text{ord}(h') = n$ ensures $h' \in \langle g \rangle^*$ in step 2, cf. Exercise 1.7.

In fact, h' is distributed uniformly on $\langle g \rangle^*$ as required.

Test $\text{ord}(h') = n$ can be evaluated efficiently, given prime factorization of n .

Expected running time of step 1 polynomial in size of n : on average

$n/\phi(n) = O(\log \log n)$ uniformly random $h' \in \langle g \rangle$ needed to find one $h' \in \langle g \rangle^*$.

Note: if n is prime, $\text{ord}(h') = n$ is equivalent to $h' \neq 1$.



Exercises

Exercise 1.31

Show (a) DH^* problem, and (b) DH^{**} problem are both random self-reducible.

Exercise 1.32

Show (a) DDH^* problem is random self-reducible, and (b) given prime factorization of n , DDH^{**} problem is random self-reducible. Hints: for both parts use three random numbers for transformation in step 1, $s, t \in_R \mathbb{Z}_n^*$ and $u \in_R \mathbb{Z}_n$; for part (b), also test for invalid inputs, cf. end of Example 1.30.

Exercises

Exercise 1.33

Show that these problems are random self-reducible for group $\langle g \rangle$ of order n :

- (a) given g^x with $x \in \mathbb{Z}_n$, compute g^{x^2} ;
- (b) given g^x with $x \in \mathbb{Z}_n^*$, compute $g^{1/x}$;
- (c) given g^x, g^y with $x, y \in \mathbb{Z}_n^*$, compute $g^{x/y}$;
- (d) given g^x, g^y with $x \in \mathbb{Z}_n$, $y \in \mathbb{Z}_n^*$, compute $g^{x/y}$;
- (e) given g^x with $x \in \mathbb{Z}_n^*$, compute g^{x^3} ;
- (f) given g^x, g^{x^2} with $x \in \mathbb{Z}_n$, compute g^{x^3} ;
- (g) given g^x, g^y with $x, y \in \mathbb{Z}_n$, compute $g^{(x+y)^2}$;
- (h) given g^x, g^y with $x, y \in \mathbb{Z}_n, x - y \in \mathbb{Z}_n^*$, compute $g^{1/(x-y)}$.

Exercises

Exercise 1.34

Let $m = pq$ be an RSA modulus, that is, p and q are large, distinct primes of bit length k , for some integer k . Let e satisfy $\gcd(e, \phi(m)) = 1$, where $\phi(m) = (p - 1)(q - 1)$. The RSA problem is to compute $x = y^{1/e} \bmod m$ given $y \in \mathbb{Z}_m^*$. Show that the RSA problem is random self-reducible.

Exercise 1.35

Let m be an RSA modulus, as in the previous exercise. Let $J_m = \{y \in \mathbb{Z}_m^* : (y/m) = 1\}$, the set of all integers in \mathbb{Z}_m^* with Jacobi symbol 1. The Quadratic Residuosity (QR) problem is to decide whether a given $y \in J_m$ is a quadratic residue modulo m or not, that is, whether $y \in QR_m$, where $QR_m = \{y \in \mathbb{Z}_m^* : \exists_{x \in \mathbb{Z}_m^*} y = x^2 \bmod m\}$. Show that the QR problem is random self-reducible.



Cryptographic Hash Functions

Definition 1.36

Function $H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a **cryptographic** hash function, if $H(x)$ is easy to compute for any x , and one or more of the following properties hold:

- **preimage resistance (onewayness):**
given y , hard to find **preimage** x such that $H(x) = y$.
- **2nd-preimage resistance (weak collision resistance):**
given x , hard to find **2nd-preimage** $x' \neq x$ such that $H(x') = H(x)$.
- **collision resistance (strong collision resistance):**
hard to find **collision** (x, x') with $x \neq x'$ such that $H(x) = H(x')$.

Well-known examples: MD5, SHA-1, SHA-256.

Collisions for MD5, SHA-0, SHA-1 and some other “old” hash functions:
[CryptographicHash-Collisions.nb](#)

Cryptographic Hash Functions

Definition 1.36

Function $H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a **cryptographic** hash function, if $H(x)$ is easy to compute for any x , and one or more of the following properties hold:

- **preimage resistance (onewayness):**
given y , hard to find **preimage** x such that $H(x) = y$.
- **2nd-preimage resistance (weak collision resistance):**
given x , hard to find **2nd-preimage** $x' \neq x$ such that $H(x') = H(x)$.
- **collision resistance (strong collision resistance):**
hard to find **collision** (x, x') with $x \neq x'$ such that $H(x) = H(x')$.

Well-known examples: MD5, SHA-1, SHA-256.

Collisions for MD5, SHA-0, SHA-1 and some other “old” hash functions:
[CryptographicHash-Collisions.nb](#)

Cryptographic Hash Functions

Definition 1.36

Function $H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a **cryptographic** hash function, if $H(x)$ is easy to compute for any x , and one or more of the following properties hold:

- **preimage resistance (onewayness):**
given y , hard to find **preimage** x such that $H(x) = y$.
- **2nd-preimage resistance (weak collision resistance):**
given x , hard to find **2nd-preimage** $x' \neq x$ such that $H(x') = H(x)$.
- **collision resistance (strong collision resistance):**
hard to find **collision** (x, x') with $x \neq x'$ such that $H(x) = H(x')$.

Well-known examples: MD5, SHA-1, SHA-256.

Collisions for MD5, SHA-0, SHA-1 and some other “old” hash functions:
[CryptographicHash-Collisions.nb](#)

Cryptographic Hash Functions

Definition 1.36

Function $H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a **cryptographic** hash function, if $H(x)$ is easy to compute for any x , and one or more of the following properties hold:

- **preimage resistance (onewayness):**
given y , hard to find **preimage** x such that $H(x) = y$.
- **2nd-preimage resistance (weak collision resistance):**
given x , hard to find **2nd-preimage** $x' \neq x$ such that $H(x') = H(x)$.
- **collision resistance (strong collision resistance):**
hard to find **collision** (x, x') with $x \neq x'$ such that $H(x) = H(x')$.

Well-known examples: MD5, SHA-1, SHA-256.

Collisions for MD5, SHA-0, SHA-1 and some other “old” hash functions:
[CryptographicHash-Collisions.nb](#)

Cryptographic Hash Functions

Definition 1.36

Function $H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a **cryptographic** hash function, if $H(x)$ is easy to compute for any x , and one or more of the following properties hold:

- **preimage resistance (onewayness):**
given y , hard to find **preimage** x such that $H(x) = y$.
- **2nd-preimage resistance (weak collision resistance):**
given x , hard to find **2nd-preimage** $x' \neq x$ such that $H(x') = H(x)$.
- **collision resistance (strong collision resistance):**
hard to find **collision** (x, x') with $x \neq x'$ such that $H(x) = H(x')$.

Well-known examples: MD5, SHA-1, SHA-256.

Collisions for MD5, SHA-0, SHA-1 and some other “old” hash functions:
[CryptographicHash-Collisions.nb](#)

Random Oracle Model

View cryptographic hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ as a random oracle.

Random oracle \mathcal{H} : **random function** of type $\{0, 1\}^* \rightarrow \{0, 1\}^k$.

We only use a **finite portion S** of \mathcal{H} as follows.

- Initially, $S = \emptyset$.
- Upon a query x :
 - if $x \notin S$, set $\mathcal{H}(x) \in_R \{0, 1\}^k$ and $S \leftarrow S \cup \{x\}$;
 - return $\mathcal{H}(x)$.

Then:

- ❶ \mathcal{H} is a **function**: same input value, same output value;
- ❷ \mathcal{H} is **random**: for different input values, output values are distributed uniformly at random (and mutually independent).

Random Oracle Model

View cryptographic hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ as a random oracle.

Random oracle \mathcal{H} : **random function** of type $\{0, 1\}^* \rightarrow \{0, 1\}^k$.

We only use a **finite portion S** of \mathcal{H} as follows.

- Initially, $S = \emptyset$.
- Upon a query x :
 - if $x \notin S$, set $\mathcal{H}(x) \in_R \{0, 1\}^k$ and $S \leftarrow S \cup \{x\}$;
 - return $\mathcal{H}(x)$.

Then:

- ➊ \mathcal{H} is a **function**: same input value, same output value;
- ➋ \mathcal{H} is **random**: for different input values, output values are distributed uniformly at random (and mutually independent).

Random Oracle Model

View cryptographic hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ as a random oracle.

Random oracle \mathcal{H} : **random function** of type $\{0, 1\}^* \rightarrow \{0, 1\}^k$.

We only use a **finite portion S** of \mathcal{H} as follows.

- Initially, $S = \emptyset$.
- Upon a query x :
 - if $x \notin S$, set $\mathcal{H}(x) \in_R \{0, 1\}^k$ and $S \leftarrow S \cup \{x\}$;
 - return $\mathcal{H}(x)$.

Then:

- ① \mathcal{H} is a **function**: same input value, same output value;
- ② \mathcal{H} is **random**: for different input values, output values are distributed uniformly at random (and mutually independent).

Random Oracle Model

View cryptographic hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ as a random oracle.

Random oracle \mathcal{H} : **random function** of type $\{0, 1\}^* \rightarrow \{0, 1\}^k$.

We only use a **finite portion S** of \mathcal{H} as follows.

- Initially, $S = \emptyset$.
- Upon a query x :
 - if $x \notin S$, set $\mathcal{H}(x) \in_R \{0, 1\}^k$ and $S \leftarrow S \cup \{x\}$;
 - return $\mathcal{H}(x)$.

Then:

- ① \mathcal{H} is a **function**: same input value, same output value;
- ② \mathcal{H} is **random**: for different input values, output values are distributed uniformly at random (and mutually independent).

Random Oracle Model

Ultimately, use of random oracle model is heuristic!

Concrete hash function such as SHA-256 is not a random function—not at all!¹

Practical upshot: protocol proved secure in random oracle model can only be broken if the attacker uses **specific properties** of hash function H .

¹Protocols do exist that can be proved secure in the random oracle model, but are insecure when used with some concrete hash function. Yet, these “counterexamples” are not realistic. See also Exercise 5.22.

Random Oracle Model

Proposition 1.37

*Let H be a cryptographic hash function, viewed as a random oracle.
Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.*

Proof. W.l.o.g. assume that all hash queries are distinct.

Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(i) On input $y \in \{0,1\}^k$, let \mathcal{E} output x . Then $\Pr[H(x) = y] \leq t/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(i) On input $y \in \{0,1\}^k$, let \mathcal{E} output x . Then $\Pr[H(x) = y] \leq t/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(i) For each hash query x , the probability that $H(x) = y$ is exactly 2^{-k} .

Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(ii) On input $x \in \{0, 1\}^*$, let \mathcal{E} output x' . Then $\Pr[x' \neq x, H(x') = H(x)] \leq t/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(ii) On input $x \in \{0, 1\}^*$, let \mathcal{E} output x' . Then $\Pr[x' \neq x, H(x') = H(x)] \leq t/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(ii) Let $y = H(x)$. For each hash query $x' \neq x$, the probability that $H(x') = y$ is exactly 2^{-k} .



Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(iii) Write $N = 2^k$ and assume $t < \sqrt{N}$. For hash queries x_1, \dots, x_t , probability of at least one collision is:

$$\begin{aligned}
 & 1 - N(N-1)(N-2) \cdots (N-t+1)/N^t \\
 &= 1 - (1 - 1/N)(1 - 2/N) \cdots (1 - (t-1)/N) \\
 &\leq 1 - e^{-2/N} e^{-4/N} \cdots e^{-2(t-1)/N} && 1-x \geq e^{-2x} \text{ for } 0 \leq x \leq 3/4 \\
 &= 1 - e^{-t(t-1)/N} \\
 &\leq 1 - (1 - t(t-1)/N) && e^x \geq 1+x \text{ for } x \in \mathbb{R} \\
 &= t(t-1)/N \\
 &\leq t^2/N.
 \end{aligned}$$



Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(iii) Write $N = 2^k$ and assume $t < \sqrt{N}$. For hash queries x_1, \dots, x_t , probability of at least one collision is:

$$\begin{aligned}
 & 1 - N(N-1)(N-2) \cdots (N-t+1)/N^t \\
 &= 1 - (1 - 1/N)(1 - 2/N) \cdots (1 - (t-1)/N) \\
 &\leq 1 - e^{-2/N} e^{-4/N} \cdots e^{-2(t-1)/N} && 1-x \geq e^{-2x} \text{ for } 0 \leq x \leq 3/4 \\
 &= 1 - e^{-t(t-1)/N} \\
 &\leq 1 - (1 - t(t-1)/N) && e^x \geq 1+x \text{ for } x \in \mathbb{R} \\
 &= t(t-1)/N \\
 &\leq t^2/N.
 \end{aligned}$$



Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(iii) Write $N = 2^k$ and assume $t < \sqrt{N}$. For hash queries x_1, \dots, x_t , probability of at least one collision is:

$$\begin{aligned}
 & 1 - N(N-1)(N-2) \cdots (N-t+1)/N^t \\
 &= 1 - (1 - 1/N)(1 - 2/N) \cdots (1 - (t-1)/N) \\
 &\leq 1 - e^{-2/N} e^{-4/N} \cdots e^{-2(t-1)/N} && 1-x \geq e^{-2x} \text{ for } 0 \leq x \leq 3/4 \\
 &= 1 - e^{-t(t-1)/N} \\
 &\leq 1 - (1 - t(t-1)/N) && e^x \geq 1+x \text{ for } x \in \mathbb{R} \\
 &= t(t-1)/N \\
 &\leq t^2/N.
 \end{aligned}$$



Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(iii) Write $N = 2^k$ and assume $t < \sqrt{N}$. For hash queries x_1, \dots, x_t , probability of at least one collision is:

$$\begin{aligned}
 & 1 - N(N-1)(N-2) \cdots (N-t+1)/N^t \\
 &= 1 - (1 - 1/N)(1 - 2/N) \cdots (1 - (t-1)/N) \\
 &\leq 1 - e^{-2/N} e^{-4/N} \cdots e^{-2(t-1)/N} && 1-x \geq e^{-2x} \text{ for } 0 \leq x \leq 3/4 \\
 &= 1 - e^{-t(t-1)/N} \\
 &\leq 1 - (1 - t(t-1)/N) && e^x \geq 1+x \text{ for } x \in \mathbb{R} \\
 &= t(t-1)/N \\
 &\leq t^2/N.
 \end{aligned}$$



Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(iii) Write $N = 2^k$ and assume $t < \sqrt{N}$. For hash queries x_1, \dots, x_t , probability of at least one collision is:

$$\begin{aligned}
 & 1 - N(N-1)(N-2) \cdots (N-t+1)/N^t \\
 &= 1 - (1 - 1/N)(1 - 2/N) \cdots (1 - (t-1)/N) \\
 &\leq 1 - e^{-2/N} e^{-4/N} \cdots e^{-2(t-1)/N} && 1-x \geq e^{-2x} \text{ for } 0 \leq x \leq 3/4 \\
 &= 1 - e^{-t(t-1)/N} \\
 &\leq 1 - (1 - t(t-1)/N) && e^x \geq 1+x \text{ for } x \in \mathbb{R} \\
 &= t(t-1)/N \\
 &\leq t^2/N.
 \end{aligned}$$



Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(iii) Write $N = 2^k$ and assume $t < \sqrt{N}$. For hash queries x_1, \dots, x_t , probability of at least one collision is:

$$\begin{aligned}
 & 1 - N(N-1)(N-2) \cdots (N-t+1)/N^t \\
 &= 1 - (1 - 1/N)(1 - 2/N) \cdots (1 - (t-1)/N) \\
 &\leq 1 - e^{-2/N} e^{-4/N} \cdots e^{-2(t-1)/N} && 1-x \geq e^{-2x} \text{ for } 0 \leq x \leq 3/4 \\
 &= 1 - e^{-t(t-1)/N} \\
 &\leq 1 - (1 - t(t-1)/N) && e^x \geq 1+x \text{ for } x \in \mathbb{R} \\
 &= t(t-1)/N \\
 &\leq t^2/N.
 \end{aligned}$$



Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(iii) Write $N = 2^k$ and assume $t < \sqrt{N}$. For hash queries x_1, \dots, x_t , probability of at least one collision is:

$$\begin{aligned}
 & 1 - N(N-1)(N-2) \cdots (N-t+1)/N^t \\
 &= 1 - (1 - 1/N)(1 - 2/N) \cdots (1 - (t-1)/N) \\
 &\leq 1 - e^{-2/N} e^{-4/N} \cdots e^{-2(t-1)/N} && 1-x \geq e^{-2x} \text{ for } 0 \leq x \leq 3/4 \\
 &= 1 - e^{-t(t-1)/N} \\
 &\leq 1 - (1 - t(t-1)/N) && e^x \geq 1+x \text{ for } x \in \mathbb{R} \\
 &= t(t-1)/N \\
 &\leq t^2/N.
 \end{aligned}$$



Random Oracle Model

Proposition 1.37

Let H be a cryptographic hash function, viewed as a random oracle.

Let (unlimitedly powerful) adversary \mathcal{E} make at most t hash queries.

(iii) Let \mathcal{E} output (x, x') . Then $\Pr[x \neq x', H(x) = H(x')] \leq t^2/2^k$.

Proof. W.l.o.g. assume that all hash queries are distinct.

(iii) Write $N = 2^k$ and assume $t < \sqrt{N}$. For hash queries x_1, \dots, x_t , probability of at least one collision is:

$$\begin{aligned}
 & 1 - N(N-1)(N-2) \cdots (N-t+1)/N^t \\
 &= 1 - (1 - 1/N)(1 - 2/N) \cdots (1 - (t-1)/N) \\
 &\leq 1 - e^{-2/N} e^{-4/N} \cdots e^{-2(t-1)/N} && 1-x \geq e^{-2x} \text{ for } 0 \leq x \leq 3/4 \\
 &= 1 - e^{-t(t-1)/N} \\
 &\leq 1 - (1 - t(t-1)/N) && e^x \geq 1+x \text{ for } x \in \mathbb{R} \\
 &= t(t-1)/N \\
 &\leq t^2/N.
 \end{aligned}$$

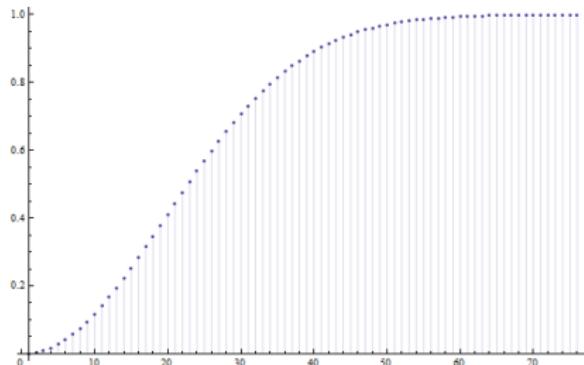


Birthday Paradox

Exercise 1.38

See the proof of Proposition 1.37(iii). Show that the upper bound for the probability of finding at least one collision is almost tight by showing that this probability is bounded below by $t(t - 1)/4N$, for $t < \sqrt{N}$.

Exact **collision probability** as function of t :



Among group of 23 people, prob. 50% that two share same birthday!
At $t = 2\sqrt{N} = 2\sqrt{2^k}$ almost surely (prob. ≈ 1) a collision.

Further Requirements for Cryptographic Hash Functions

Any deviation from what statistically holds for random functions should be:

- absent,
- unlikely, or
- in any case, infeasible to exploit.

Example (Partial preimage resistance (or, local onewayness))

Given y , hard to find (partial) information about any preimage x s.t. $H(x) = y$.

Exercise 1.39

Let H be a preimage resistant hash function. Show that partial preimage resistance is strictly stronger than preimage resistance, by constructing a preimage resistant hash function H' (from H) which is not partial preimage resistant.

Further Requirements for Cryptographic Hash Functions

Any deviation from what statistically holds for random functions should be:

- absent,
- unlikely, or
- in any case, infeasible to exploit.

Example (Partial preimage resistance (or, local onewayness))

Given y , hard to find (partial) information about any preimage x s.t. $H(x) = y$.

Exercise 1.39

Let H be a preimage resistant hash function. Show that partial preimage resistance is strictly stronger than preimage resistance, by constructing a preimage resistant hash function H' (from H) which is not partial preimage resistant.

Further Requirements for Cryptographic Hash Functions

Any deviation from what statistically holds for random functions should be:

- absent,
- unlikely, or
- in any case, infeasible to exploit.

Example (Partial preimage resistance (or, local onewayness))

Given y , hard to find (partial) information about any preimage x s.t. $H(x) = y$.

Exercise 1.39

Let H be a preimage resistant hash function. Show that partial preimage resistance is strictly stronger than preimage resistance, by constructing a preimage resistant hash function H' (from H) which is not partial preimage resistant.



Exercises

Exercise 1.40

Suppose one demands of a hash function H that it is hard to find a pair of bit strings (x, x') satisfying $H(x) = \bar{H}(x')$, where \bar{s} denotes the bitwise complement of bit string s . Analyze the probability that an adversary \mathcal{E} making at most t hash queries finds such a pair, where H is viewed as a random oracle.

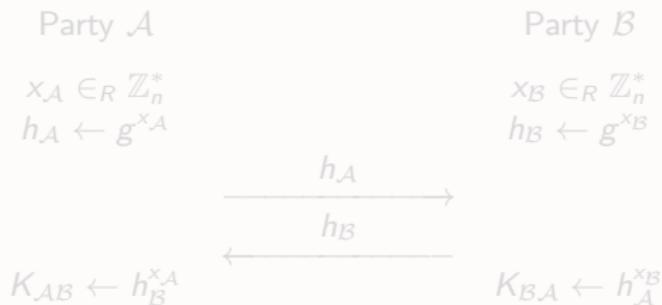
Exercise 1.41

Let $y = H^2(x)$, where $x \in \{0, 1\}^*$, and assume that $y \neq H(x)$. Let \mathcal{E} be an adversary that, given y only, makes t hash queries on distinct inputs $x_1, \dots, x_t \in \{0, 1\}^k \setminus \{y\}$. View $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ as a random oracle to show that \mathcal{E} finds a preimage of y with probability exactly equal to $\epsilon(2 - \epsilon)$, with $\epsilon = t/2^k$. Also, argue why there is no contradiction with Proposition 1.37(i) even though $\epsilon(2 - \epsilon) \geq \epsilon$.



Diffie-Hellman Key Exchange (1976)

Discrete Log setting: finite cyclic group $\langle g \rangle$ of order n .



Key $K = K_{AB} = K_{BA}$ is a **shared key for A and B**, meaning that:

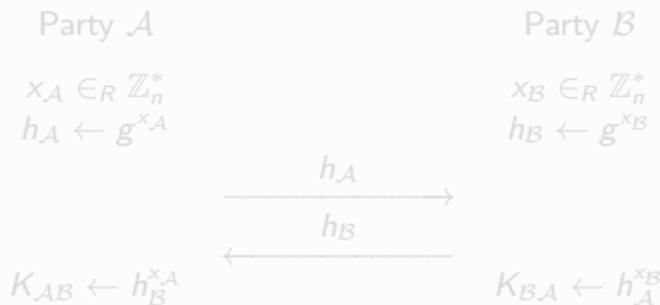
- (i) K is the same for A and B
- (ii) K is *private*, only known to A and B
- (iii) K is actually equal to $g^{x_A x_B}$ (with contributions from both parties)

Often, key K is subsequently used as a session key.

How “**secure**” is this protocol?

Diffie-Hellman Key Exchange (1976)

Discrete Log setting: finite cyclic group $\langle g \rangle$ of order n .



Key $K = K_{AB} = K_{BA}$ is a **shared key for A and B**, meaning that:

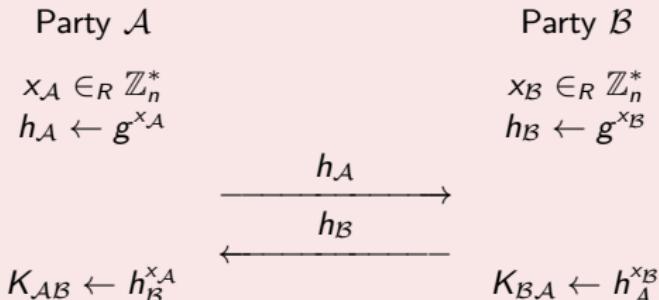
- (i) K is the same for A and B
- (ii) K is *private*, only known to A and B
- (iii) K is actually equal to $g^{x_A x_B}$ (with contributions from both parties)

Often, key K is subsequently used as a session key.

How “**secure**” is this protocol?

Diffie-Hellman Key Exchange (1976)

Discrete Log setting: finite cyclic group $\langle g \rangle$ of order n .



Key $K = K_{AB} = K_{BA}$ is a **shared key for A and B**, meaning that:

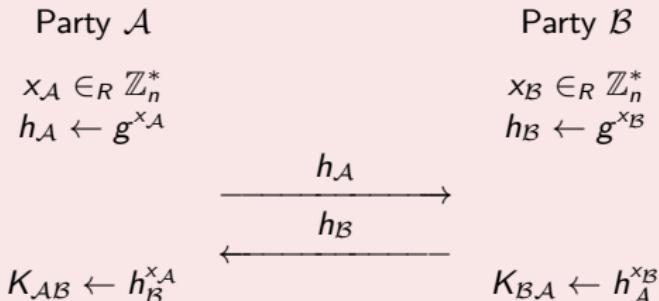
- (i) K is the same for A and B
- (ii) K is *private*, only known to A and B
- (iii) K is actually equal to $g^{x_A x_B}$ (with contributions from both parties)

Often, key K is subsequently used as a session key.

How “**secure**” is this protocol?

Diffie-Hellman Key Exchange (1976)

Discrete Log setting: finite cyclic group $\langle g \rangle$ of order n .



Key $K = K_{AB} = K_{BA}$ is a **shared key for A and B**, meaning that:

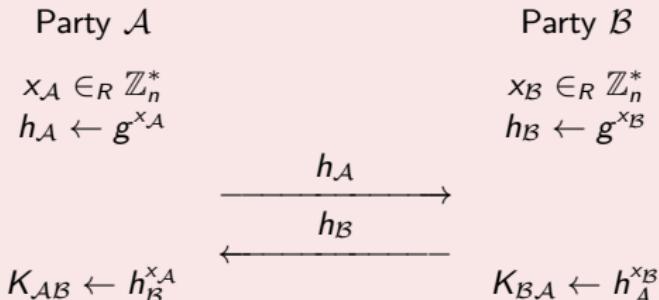
- (i) K is the same for A and B
- (ii) K is *private*, only known to A and B
- (iii) K is actually equal to $g^{x_A x_B}$ (with contributions from both parties)

Often, key K is subsequently used as a session key.

How “**secure**” is this protocol?

Diffie-Hellman Key Exchange (1976)

Discrete Log setting: finite cyclic group $\langle g \rangle$ of order n .



Key $K = K_{AB} = K_{BA}$ is a **shared key for A and B**, meaning that:

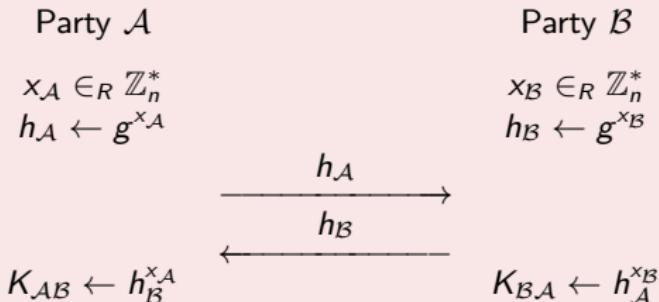
- (i) K is the same for A and B
- (ii) K is *private*, only known to A and B
- (iii) K is actually equal to $g^{x_A x_B}$ (with contributions from both parties)

Often, key K is subsequently used as a session key.

How “**secure**” is this protocol?

Diffie-Hellman Key Exchange (1976)

Discrete Log setting: finite cyclic group $\langle g \rangle$ of order n .



Key $K = K_{AB} = K_{BA}$ is a **shared key for A and B**, meaning that:

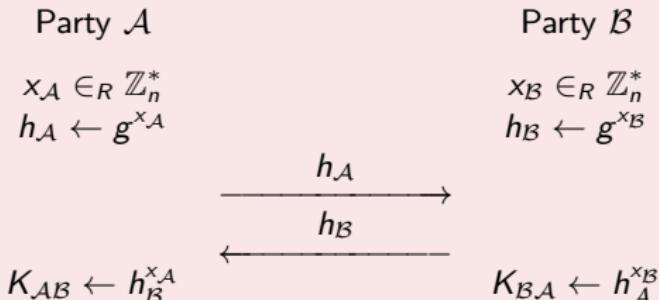
- (i) K is the same for A and B
- (ii) K is *private*, only known to A and B
- (iii) K is actually equal to $g^{x_A x_B}$ (with contributions from both parties)

Often, key K is subsequently used as a session key.

How “**secure**” is this protocol?

Diffie-Hellman Key Exchange (1976)

Discrete Log setting: finite cyclic group $\langle g \rangle$ of order n .



Key $K = K_{AB} = K_{BA}$ is a **shared key for A and B**, meaning that:

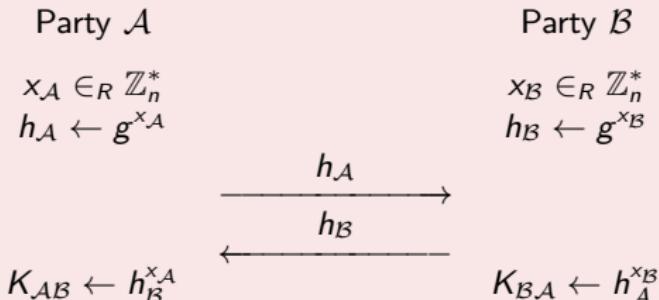
- (i) K is the same for A and B
- (ii) K is *private*, only known to A and B
- (iii) K is actually equal to $g^{x_A x_B}$ (with contributions from both parties)

Often, key K is subsequently used as a session key.

How “**secure**” is this protocol?

Diffie-Hellman Key Exchange (1976)

Discrete Log setting: finite cyclic group $\langle g \rangle$ of order n .



Key $K = K_{AB} = K_{BA}$ is a **shared key for A and B**, meaning that:

- (i) K is the same for A and B
- (ii) K is *private*, only known to A and B
- (iii) K is actually equal to $g^{x_A x_B}$ (with contributions from both parties)

Often, key K is subsequently used as a **session key**.

How “**secure**” is this protocol?

Exercise 2.1

Explain what happens if the secret exponents x_A and x_B are chosen from \mathbb{Z}_n instead of \mathbb{Z}_n^ . Confirm your findings by computing the statistical distance $\Delta(K; K')$, where $K = g^{x_A x_B}$ with $x_A, x_B \in_R \mathbb{Z}_n^*$ and $K' = g^{x'_A x'_B}$ with $x'_A, x'_B \in_R \mathbb{Z}_n$.*

Security against Passive Attacks

Passive attacker \mathcal{E} eavesdrops on communication,
gets $h_A = g^{x_A}$ and $h_B = g^{x_B}$.
 \mathcal{E} knows the DL setting in use.
Thus, \mathcal{E} may try to compute $x_A = \log_g h_A$.

But, success would contradict the **DL assumption** (Definition 1.19).

Similarly, computation of $x_B = \log_g h_B$ cannot be successful.

Can we make this line of reasoning rigorous?

Security against Passive Attacks

Passive attacker \mathcal{E} eavesdrops on communication,
gets $h_A = g^{x_A}$ and $h_B = g^{x_B}$.
 \mathcal{E} knows the DL setting in use.
Thus, \mathcal{E} may try to compute $x_A = \log_g h_A$.

But, success would contradict the **DL assumption** (Definition 1.19).

Similarly, computation of $x_B = \log_g h_B$ cannot be successful.

Can we make this line of reasoning rigorous?

Security against Passive Attacks

Passive attacker \mathcal{E} eavesdrops on communication,
gets $h_A = g^{x_A}$ and $h_B = g^{x_B}$.
 \mathcal{E} knows the DL setting in use.
Thus, \mathcal{E} may try to compute $x_A = \log_g h_A$.

But, success would contradict the **DL assumption** (Definition 1.19).

Similarly, computation of $x_B = \log_g h_B$ cannot be successful.

Can we make this line of reasoning rigorous?

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ➊ put $h_A = h$
- ➋ put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ➌ let \mathcal{E} perform its attack
- ➍ upon success, \mathcal{E} outputs $g^{xx'}$,
- ➎ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ➊ put $h_A = h$
- ➋ put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ➌ let \mathcal{E} perform its attack
- ➍ upon success, \mathcal{E} outputs $g^{xx'}$,
- ➎ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ➊ put $h_A = h$
- ➋ put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ➌ let \mathcal{E} perform its attack
- ➍ upon success, \mathcal{E} outputs $g^{xx'}$,
- ➎ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ➊ put $h_A = h$
- ➋ put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ➌ let \mathcal{E} perform its attack
- ➍ upon success, \mathcal{E} outputs $g^{xx'}$,
- ➎ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ① put $h_A = h$
- ② put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ③ let \mathcal{E} perform its attack
- ④ upon success, \mathcal{E} outputs $g^{xx'}$.
- ⑤ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ➊ put $h_A = h$
- ➋ put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ➌ let \mathcal{E} perform its attack
- ➍ upon success, \mathcal{E} outputs $g^{xx'}$.
- ➎ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ① put $h_A = h$
- ② put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ③ let \mathcal{E} perform its attack
- ④ upon success, \mathcal{E} outputs $g^{xx'}$.
- ⑤ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ① put $h_A = h$
- ② put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ③ let \mathcal{E} perform its attack
- ④ upon success, \mathcal{E} outputs $g^{xx'}$.
- ⑤ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ① put $h_A = h$
- ② put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ③ let \mathcal{E} perform its attack
- ④ upon success, \mathcal{E} outputs $g^{xx'}$.
- ⑤ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ① put $h_A = h$
- ② put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ③ let \mathcal{E} perform its attack
- ④ upon success, \mathcal{E} outputs $g^{xx'}$.
- ⑤ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ① put $h_A = h$
- ② put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ③ let \mathcal{E} perform its attack
- ④ upon success, \mathcal{E} outputs $g^{xx'}$.
- ⑤ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Security Proof by Reduction

Reduction:

If attacker \mathcal{E} can find K from h_A and h_B ,
 then we can use \mathcal{E} to solve the DL problem.

Given an instance of the DL problem: $h = g^x$.

How to use \mathcal{E} to find $x = \log_g h$?

Attempt at reduction:

- ① put $h_A = h$
- ② put $h_B = g^{x'}$ with $x' \in_R \mathbb{Z}_n^*$
- ③ let \mathcal{E} perform its attack
- ④ upon success, \mathcal{E} outputs $g^{xx'}$.
- ⑤ ... ?

How to find x from $g^{xx'}$? Even knowing x' .

Way out: introduce **DH assumption** (Definition 1.20).

Enough Security?

Under DH assumption, attacker \mathcal{E} cannot find key K given h_A and h_B . But, suppose K is used to encrypt 1-bit message $M \in \{0, 1\}$: ciphertext $C = g^M K$.

Example 2.2

Let $\langle g \rangle = \mathbb{Z}_p^*$, with $p' = (p - 1)/2$ prime. Then $\mathbb{Z}_p^* = QR_p \cup \overline{QR_p}$ with

$$\begin{aligned} QR_p &= \{1, g^2, \dots, g^{2p'-2}\} \\ \overline{QR_p} &= \{g, g^3, \dots, g^{2p'-1}\} \end{aligned}$$

quadratic residues mod p
quadratic non-residues mod p

h_A	h_B	K	
QR_p	QR_p	QR_p	25%
QR_p	$\overline{QR_p}$	QR_p	25%
$\overline{QR_p}$	QR_p	QR_p	25%
$\overline{QR_p}$	$\overline{QR_p}$	QR_p	25%

Assume $M \in_R \{0, 1\}$.
Given $C = g^M K$.

Guess: $\hat{M} = \begin{cases} 0, & \text{if } C \in QR_p, \\ 1, & \text{if } C \in \overline{QR_p}. \end{cases}$

$\Pr[\hat{M} = M] = 3/4$ (check this!)

Even worse: M can be recovered deterministically from the quadratic residuosity of h_A , h_B , and C . See Exercise 2.6.

Enough Security?

Under DH assumption, attacker \mathcal{E} cannot find key K given h_A and h_B . But, suppose K is used to encrypt 1-bit message $M \in \{0, 1\}$: ciphertext $C = g^M K$.

Example 2.2

Let $\langle g \rangle = \mathbb{Z}_p^*$, with $p' = (p - 1)/2$ prime. Then $\mathbb{Z}_p^* = QR_p \cup \overline{QR_p}$ with

$$\begin{aligned} QR_p &= \{1, g^2, \dots, g^{2p'-2}\} \\ \overline{QR_p} &= \{g, g^3, \dots, g^{2p'-1}\} \end{aligned}$$

quadratic residues mod p
quadratic non-residues mod p

h_A	h_B	K	
QR_p	QR_p	QR_p	25%
QR_p	$\overline{QR_p}$	QR_p	25%
$\overline{QR_p}$	QR_p	QR_p	25%
$\overline{QR_p}$	$\overline{QR_p}$	QR_p	25%

Assume $M \in_R \{0, 1\}$.
Given $C = g^M K$.

Guess: $\hat{M} = \begin{cases} 0, & \text{if } C \in QR_p, \\ 1, & \text{if } C \in \overline{QR_p}. \end{cases}$

$\Pr[\hat{M} = M] = 3/4$ (check this!)

Even worse: M can be recovered deterministically from the quadratic residuosity of h_A , h_B , and C . See Exercise 2.6.

Enough Security?

Under DH assumption, attacker \mathcal{E} cannot find key K given h_A and h_B . But, suppose K is used to encrypt 1-bit message $M \in \{0, 1\}$: ciphertext $C = g^M K$.

Example 2.2

Let $\langle g \rangle = \mathbb{Z}_p^*$, with $p' = (p - 1)/2$ prime. Then $\mathbb{Z}_p^* = QR_p \cup \overline{QR_p}$ with

$$\begin{array}{ll} QR_p = \{1, g^2, \dots, g^{2p'-2}\} & \text{quadratic residues mod } p \\ \overline{QR_p} = \{g, g^3, \dots, g^{2p'-1}\} & \text{quadratic non-residues mod } p \end{array}$$

h_A	h_B	K
QR_p	QR_p	QR_p 25%
QR_p	$\overline{QR_p}$	QR_p 25%
$\overline{QR_p}$	QR_p	QR_p 25%
$\overline{QR_p}$	$\overline{QR_p}$	QR_p 25%

Assume $M \in_R \{0, 1\}$.
 Given $C = g^M K$.

Guess: $\hat{M} = \begin{cases} 0, & \text{if } C \in QR_p, \\ 1, & \text{if } C \in \overline{QR_p}. \end{cases}$

$\Pr[\hat{M} = M] = 3/4$ (check this!)

Even worse: M can be recovered deterministically from the quadratic residuosity of h_A , h_B , and C . See Exercise 2.6.

Enough Security?

Under DH assumption, attacker \mathcal{E} cannot find key K given h_A and h_B . But, suppose K is used to encrypt 1-bit message $M \in \{0, 1\}$: ciphertext $C = g^M K$.

Example 2.2

Let $\langle g \rangle = \mathbb{Z}_p^*$, with $p' = (p - 1)/2$ prime. Then $\mathbb{Z}_p^* = QR_p \cup \overline{QR_p}$ with

$$\begin{array}{ll} QR_p = \{1, g^2, \dots, g^{2p'-2}\} & \text{quadratic residues mod } p \\ \overline{QR_p} = \{g, g^3, \dots, g^{2p'-1}\} & \text{quadratic non-residues mod } p \end{array}$$

h_A	h_B	K
QR_p	QR_p	QR_p 25%
QR_p	$\overline{QR_p}$	QR_p 25%
$\overline{QR_p}$	QR_p	QR_p 25%
$\overline{QR_p}$	$\overline{QR_p}$	$\overline{QR_p}$ 25%

Assume $M \in_R \{0, 1\}$.
 Given $C = g^M K$.

Guess: $\hat{M} = \begin{cases} 0, & \text{if } C \in QR_p, \\ 1, & \text{if } C \in \overline{QR_p}. \end{cases}$

$\Pr[\hat{M} = M] = 3/4$ (check this!)

Even worse: M can be recovered deterministically from the quadratic residuosity of h_A , h_B , and C . See Exercise 2.6.

Enough Security?

Under DH assumption, attacker \mathcal{E} cannot find key K given h_A and h_B . But, suppose K is used to encrypt 1-bit message $M \in \{0, 1\}$: ciphertext $C = g^M K$.

Example 2.2

Let $\langle g \rangle = \mathbb{Z}_p^*$, with $p' = (p - 1)/2$ prime. Then $\mathbb{Z}_p^* = QR_p \cup \overline{QR_p}$ with

$$\begin{aligned} QR_p &= \{1, g^2, \dots, g^{2p'-2}\} && \text{quadratic residues mod } p \\ \overline{QR_p} &= \{g, g^3, \dots, g^{2p'-1}\} && \text{quadratic non-residues mod } p \end{aligned}$$

h_A	h_B	K
QR_p	QR_p	QR_p 25%
QR_p	$\overline{QR_p}$	QR_p 25%
$\overline{QR_p}$	QR_p	QR_p 25%
$\overline{QR_p}$	$\overline{QR_p}$	QR_p 25%

Assume $M \in_R \{0, 1\}$.
 Given $C = g^M K$.

Guess: $\hat{M} = \begin{cases} 0, & \text{if } C \in QR_p, \\ 1, & \text{if } C \in \overline{QR_p}. \end{cases}$

$\Pr[\hat{M} = M] = 3/4$ (check this!)

Even worse: M can be recovered deterministically from the quadratic residuosity of h_A , h_B , and C . See Exercise 2.6.

Enough Security?

Under DH assumption, attacker \mathcal{E} cannot find key K given h_A and h_B . But, suppose K is used to encrypt 1-bit message $M \in \{0, 1\}$: ciphertext $C = g^M K$.

Example 2.2

Let $\langle g \rangle = \mathbb{Z}_p^*$, with $p' = (p - 1)/2$ prime. Then $\mathbb{Z}_p^* = QR_p \cup \overline{QR_p}$ with

$$\begin{aligned} QR_p &= \{1, g^2, \dots, g^{2p'-2}\} && \text{quadratic residues mod } p \\ \overline{QR_p} &= \{g, g^3, \dots, g^{2p'-1}\} && \text{quadratic non-residues mod } p \end{aligned}$$

h_A	h_B	K
QR_p	QR_p	QR_p 25%
QR_p	$\overline{QR_p}$	QR_p 25%
$\overline{QR_p}$	QR_p	QR_p 25%
$\overline{QR_p}$	$\overline{QR_p}$	QR_p 25%

Assume $M \in_R \{0, 1\}$.
 Given $C = g^M K$.

Guess: $\hat{M} = \begin{cases} 0, & \text{if } C \in QR_p, \\ 1, & \text{if } C \in \overline{QR_p}. \end{cases}$

$\Pr[\hat{M} = M] = 3/4$ (check this!)

Even worse: M can be recovered deterministically from the quadratic residuosity of h_A , h_B , and C . See Exercise 2.6.



Enough Security?

Under DH assumption, attacker \mathcal{E} cannot find key K given h_A and h_B . But, suppose K is used to encrypt 1-bit message $M \in \{0, 1\}$: ciphertext $C = g^M K$.

Example 2.2

Let $\langle g \rangle = \mathbb{Z}_p^*$, with $p' = (p - 1)/2$ prime. Then $\mathbb{Z}_p^* = QR_p \cup \overline{QR_p}$ with

$$\begin{aligned} QR_p &= \{1, g^2, \dots, g^{2p'-2}\} && \text{quadratic residues mod } p \\ \overline{QR_p} &= \{g, g^3, \dots, g^{2p'-1}\} && \text{quadratic non-residues mod } p \end{aligned}$$

h_A	h_B	K
QR_p	QR_p	QR_p 25%
QR_p	$\overline{QR_p}$	QR_p 25%
$\overline{QR_p}$	QR_p	QR_p 25%
$\overline{QR_p}$	$\overline{QR_p}$	QR_p 25%

Assume $M \in_R \{0, 1\}$.
 Given $C = g^M K$.

Guess: $\hat{M} = \begin{cases} 0, & \text{if } C \in QR_p, \\ 1, & \text{if } C \in \overline{QR_p}. \end{cases}$

$\Pr[\hat{M} = M] = 3/4$ (check this!)

Even worse: M can be recovered deterministically from the quadratic residuosity of h_A , h_B , and C . See Exercise 2.6.



Choice for Prime Order Groups

DDH assumption (Definition 1.21) to exclude any leakage of *partial information*.

Exercise 2.3

Argue that the DDH assumption is false when n contains a small prime factor.

Simple way to exclude small prime factors: let n be **sufficiently large prime**.

Additional benefit: \mathbb{Z}_n is a field!

Security under DDH Assumption

Balanced function $f : \langle g \rangle \rightarrow \{0, 1\}$, if *a priori*

$\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \langle g \rangle$.

Suppose \mathcal{E} predicts $f(K)$ better than guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K)] > 1/2 + \epsilon, \quad \text{for non-negligible } \epsilon$$

Given \mathcal{E} , define distinguisher D for DDH:

$$D(g^x, g^y, g^z) := \begin{cases} 1, & \text{if } \mathcal{E}(g^x, g^y) = f(g^z), \\ 0, & \text{if } \mathcal{E}(g^x, g^y) \neq f(g^z). \end{cases}$$

On **random triples**:

On **DH-triples**:

$$\begin{aligned} & \Pr[D(h_A, h_B, K) = 1] \\ &= \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ &> 1/2 + \epsilon \end{aligned}$$

$$\begin{aligned} & \Pr[D(h_A, h_B, g^z) = 1] \\ &= \Pr[\mathcal{E}(h_A, h_B) = f(g^z)] \\ & \quad \text{"}f \text{ is balanced"} \\ &= 1/2 \end{aligned}$$

Advantage of D exceeds ϵ . Contradicts DDH assumption.

Security under DDH Assumption

Balanced function $f : \langle g \rangle \rightarrow \{0, 1\}$, if *a priori*

$\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \langle g \rangle$.

Suppose \mathcal{E} predicts $f(K)$ better than guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K)] > 1/2 + \epsilon, \quad \text{for non-negligible } \epsilon$$

Given \mathcal{E} , define distinguisher D for DDH:

$$D(g^x, g^y, g^z) := \begin{cases} 1, & \text{if } \mathcal{E}(g^x, g^y) = f(g^z), \\ 0, & \text{if } \mathcal{E}(g^x, g^y) \neq f(g^z). \end{cases}$$

On **random triples**:

On **DH-triples**:

$$\begin{aligned} & \Pr[D(h_A, h_B, K) = 1] \\ &= \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ &> 1/2 + \epsilon \end{aligned}$$

$$\begin{aligned} & \Pr[D(h_A, h_B, g^z) = 1] \\ &= \Pr[\mathcal{E}(h_A, h_B) = f(g^z)] \\ & \quad \text{"}f \text{ is balanced"} \\ &= 1/2 \end{aligned}$$

Advantage of D exceeds ϵ . Contradicts DDH assumption.

Security under DDH Assumption

Balanced function $f : \langle g \rangle \rightarrow \{0, 1\}$, if *a priori*

$\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \langle g \rangle$.

Suppose \mathcal{E} predicts $f(K)$ better than guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K)] > 1/2 + \epsilon, \quad \text{for non-negligible } \epsilon$$

Given \mathcal{E} , define distinguisher D for DDH:

$$D(g^x, g^y, g^z) := \begin{cases} 1, & \text{if } \mathcal{E}(g^x, g^y) = f(g^z), \\ 0, & \text{if } \mathcal{E}(g^x, g^y) \neq f(g^z). \end{cases}$$

On **random triples**:

On **DH-triples**:

$$\begin{aligned} & \Pr[D(h_A, h_B, K) = 1] \\ &= \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ &> 1/2 + \epsilon \end{aligned}$$

$$\begin{aligned} & \Pr[D(h_A, h_B, g^z) = 1] \\ &= \Pr[\mathcal{E}(h_A, h_B) = f(g^z)] \\ & \quad \text{"f is balanced"} \\ &= 1/2 \end{aligned}$$

Advantage of D exceeds ϵ . Contradicts DDH assumption.

Security under DDH Assumption

Balanced function $f : \langle g \rangle \rightarrow \{0, 1\}$, if *a priori*

$\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \langle g \rangle$.

Suppose \mathcal{E} predicts $f(K)$ better than guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K)] > 1/2 + \epsilon, \quad \text{for non-negligible } \epsilon$$

Given \mathcal{E} , define distinguisher D for DDH:

$$D(g^x, g^y, g^z) := \begin{cases} 1, & \text{if } \mathcal{E}(g^x, g^y) = f(g^z), \\ 0, & \text{if } \mathcal{E}(g^x, g^y) \neq f(g^z). \end{cases}$$

On **random triples**:

On **DH-triples**:

$$\begin{aligned} & \Pr[D(h_A, h_B, K) = 1] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ > & 1/2 + \epsilon \end{aligned}$$

$$\begin{aligned} & \Pr[D(h_A, h_B, g^z) = 1] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(g^z)] \\ & \quad "f \text{ is balanced}" \\ = & 1/2 \end{aligned}$$

Advantage of D exceeds ϵ . Contradicts DDH assumption.

Security under DDH Assumption

Balanced function $f : \langle g \rangle \rightarrow \{0, 1\}$, if *a priori*

$\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \langle g \rangle$.

Suppose \mathcal{E} predicts $f(K)$ better than guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K)] > 1/2 + \epsilon, \quad \text{for non-negligible } \epsilon$$

Given \mathcal{E} , define distinguisher D for DDH:

$$D(g^x, g^y, g^z) := \begin{cases} 1, & \text{if } \mathcal{E}(g^x, g^y) = f(g^z), \\ 0, & \text{if } \mathcal{E}(g^x, g^y) \neq f(g^z). \end{cases}$$

On **random triples**:

On **DH-triples**:

$$\begin{aligned} & \Pr[D(h_A, h_B, K) = 1] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ > & 1/2 + \epsilon \end{aligned}$$

$$\begin{aligned} & \Pr[D(h_A, h_B, g^z) = 1] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(g^z)] \\ & \quad "f \text{ is balanced}" \\ = & 1/2 \end{aligned}$$

Advantage of D exceeds ϵ . Contradicts DDH assumption.

Security under DDH Assumption

Balanced function $f : \langle g \rangle \rightarrow \{0, 1\}$, if *a priori*

$\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \langle g \rangle$.

Suppose \mathcal{E} predicts $f(K)$ better than guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K)] > 1/2 + \epsilon, \quad \text{for non-negligible } \epsilon$$

Given \mathcal{E} , define distinguisher D for DDH:

$$D(g^x, g^y, g^z) := \begin{cases} 1, & \text{if } \mathcal{E}(g^x, g^y) = f(g^z), \\ 0, & \text{if } \mathcal{E}(g^x, g^y) \neq f(g^z). \end{cases}$$

On **random triples**:

On **DH-triples**:

$$\begin{aligned} & \Pr[D(h_A, h_B, K) = 1] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ > & 1/2 + \epsilon \end{aligned}$$

$$\begin{aligned} & \Pr[D(h_A, h_B, g^z) = 1] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(g^z)] \\ & \quad "f \text{ is balanced}" \\ = & 1/2 \end{aligned}$$

Advantage of D exceeds ϵ . Contradicts DDH assumption.

2.1 Diffie-Hellman Key Exchange
2.2 Authenticated Key Exchange

2.1.1 Basic Protocol
2.1.2 Passive Attacks
2.1.3 A Practical Variant
2.1.4 Aside: ElGamal Encryption

Exercise

Exercise 2.4

Extend the above analysis to the case that the a priori probabilities are given by $\Pr[f(u) = 0] = p_0$ and $\Pr[f(u) = 1] = p_1 = 1 - p_0$.

Hashed DH Key Exchange

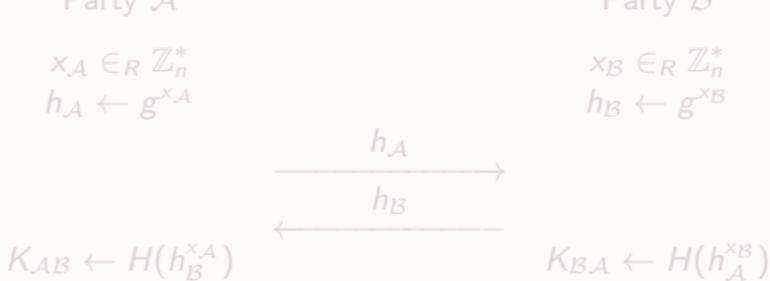
Cryptographic hash function $H : \langle g \rangle \rightarrow \{0, 1\}^k$.

Party \mathcal{A}

$$\begin{aligned} x_{\mathcal{A}} &\in_R \mathbb{Z}_n^* \\ h_{\mathcal{A}} &\leftarrow g^{x_{\mathcal{A}}} \end{aligned}$$

Party \mathcal{B}

$$\begin{aligned} x_{\mathcal{B}} &\in_R \mathbb{Z}_n^* \\ h_{\mathcal{B}} &\leftarrow g^{x_{\mathcal{B}}} \end{aligned}$$



$K = H(g^{x_A x_B})$ is a k -bit approximately uniformly random key.
 Assuming $n \gg 2^k$ (e.g., $n \approx 2^{256}$ and $k = 128$).

Hashed DH Key Exchange

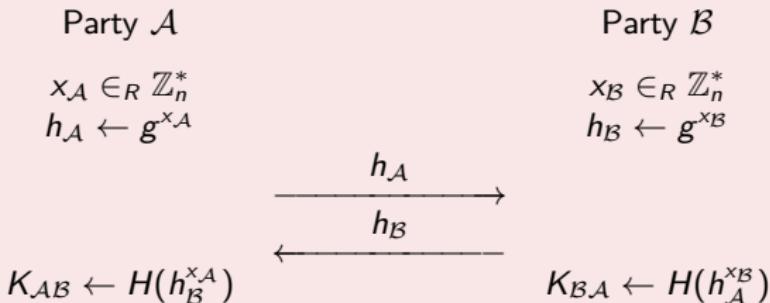
Cryptographic hash function $H : \langle g \rangle \rightarrow \{0, 1\}^k$.

Party \mathcal{A}

$$\begin{aligned} x_{\mathcal{A}} &\in_R \mathbb{Z}_n^* \\ h_{\mathcal{A}} &\leftarrow g^{x_{\mathcal{A}}} \end{aligned}$$

Party \mathcal{B}

$$\begin{aligned} x_{\mathcal{B}} &\in_R \mathbb{Z}_n^* \\ h_{\mathcal{B}} &\leftarrow g^{x_{\mathcal{B}}} \end{aligned}$$



$K = H(g^{x_{\mathcal{A}}x_{\mathcal{B}}})$ is a k -bit approximately uniformly random key.
 Assuming $n \gg 2^k$ (e.g., $n \approx 2^{256}$ and $k = 128$).

Hashed DH Key Exchange

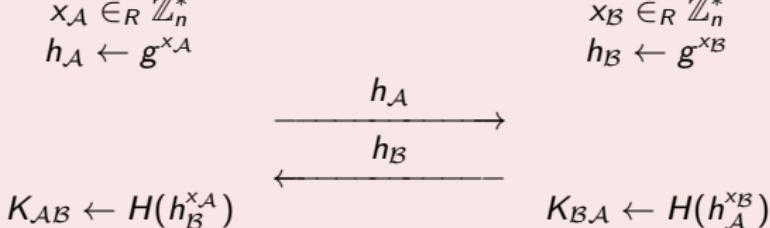
Cryptographic hash function $H : \langle g \rangle \rightarrow \{0, 1\}^k$.

Party \mathcal{A}

$$\begin{aligned} x_{\mathcal{A}} &\in_R \mathbb{Z}_n^* \\ h_{\mathcal{A}} &\leftarrow g^{x_{\mathcal{A}}} \end{aligned}$$

Party \mathcal{B}

$$\begin{aligned} x_{\mathcal{B}} &\in_R \mathbb{Z}_n^* \\ h_{\mathcal{B}} &\leftarrow g^{x_{\mathcal{B}}} \end{aligned}$$



$K = H(g^{x_A x_B})$ is a k -bit approximately uniformly random key.
 Assuming $n \gg 2^k$ (e.g., $n \approx 2^{256}$ and $k = 128$).

Security Proof

Balanced function $f : \{0, 1\}^k \rightarrow \{0, 1\}$, if *a priori*

$$\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2 \text{ for } u \in_R \{0, 1\}^k.$$

Intuition is to consider “lucky” event L : “ \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$ ”

If $\neg L$, then \mathcal{E} has no advantage over guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] = 1/2$$

If L , then \mathcal{E} might be successful, so use trivial upper bound:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \leq 1$$

Hence:

$$\begin{aligned} & \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ \leq & \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ = & \Pr[L] + \frac{1}{2}(1 - \Pr[L]) \\ = & \frac{1}{2} + \frac{1}{2} \Pr[L]. \end{aligned}$$

Security Proof

Balanced function $f : \{0, 1\}^k \rightarrow \{0, 1\}$, if *a priori*
 $\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \{0, 1\}^k$.

Intuition is to consider “**lucky**” event L : “ \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$ ”

If $\neg L$, then \mathcal{E} has no advantage over guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] = 1/2$$

If L , then \mathcal{E} might be successful, so use trivial upper bound:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \leq 1$$

Hence:

$$\begin{aligned} & \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ \leq & \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ = & \Pr[L] + \frac{1}{2}(1 - \Pr[L]) \\ = & \frac{1}{2} + \frac{1}{2} \Pr[L]. \end{aligned}$$

Security Proof

Balanced function $f : \{0, 1\}^k \rightarrow \{0, 1\}$, if *a priori*
 $\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \{0, 1\}^k$.

Intuition is to consider “**lucky**” event L : “ \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$ ”

If $\neg L$, then \mathcal{E} has no advantage over guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] = 1/2$$

If L , then \mathcal{E} might be successful, so use trivial upper bound:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \leq 1$$

Hence:

$$\begin{aligned} & \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ = & \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ \leq & \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ = & \Pr[L] + \frac{1}{2}(1 - \Pr[L]) \\ = & \frac{1}{2} + \frac{1}{2} \Pr[L]. \end{aligned}$$

Security Proof

Balanced function $f : \{0, 1\}^k \rightarrow \{0, 1\}$, if *a priori*
 $\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2$ for $u \in_R \{0, 1\}^k$.

Intuition is to consider “**lucky**” event L : “ \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$ ”

If $\neg L$, then \mathcal{E} has no advantage over guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] = 1/2$$

If L , then \mathcal{E} might be successful, so use trivial upper bound:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \leq 1$$

Hence:

$$\begin{aligned} & \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ &= \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ &\leq \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ &= \Pr[L] + \frac{1}{2}(1 - \Pr[L]) \\ &= \frac{1}{2} + \frac{1}{2} \Pr[L]. \end{aligned}$$

Security Proof

Balanced function $f : \{0, 1\}^k \rightarrow \{0, 1\}$, if *a priori*

$$\Pr[f(u) = 0] = \Pr[f(u) = 1] = 1/2 \text{ for } u \in_R \{0, 1\}^k.$$

Intuition is to consider “**lucky**” event L : “ \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$ ”

If $\neg L$, then \mathcal{E} has no advantage over guessing at random:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] = 1/2$$

If L , then \mathcal{E} might be successful, so use trivial upper bound:

$$\Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \leq 1$$

Hence:

$$\begin{aligned} & \Pr[\mathcal{E}(h_A, h_B) = f(K)] \\ &= \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid L] \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ &\leq \Pr[L] + \Pr[\mathcal{E}(h_A, h_B) = f(K) \mid \neg L] \Pr[\neg L] \\ &= \Pr[L] + \frac{1}{2}(1 - \Pr[L]) \\ &= \frac{1}{2} + \frac{1}{2} \Pr[L]. \end{aligned}$$

Security Proof: Bounding $\Pr[L]$

Event L : during attack, \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$.

We show that $\Pr[L]$ is negligible under the DH assumption.

Define p.p.t. algorithm \mathcal{E}' , using p.p.t. \mathcal{E} as a subroutine:

- ➊ take h_A and h_B as inputs;
- ➋ run \mathcal{E} on these inputs, while recording all \mathcal{H} queries made by \mathcal{E} ;
- ➌ return an \mathcal{H} query (made by \mathcal{E}) at random.

Let N denote total number of \mathcal{H} queries. Then:

$$\begin{aligned} \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}] &= \Pr[L]/N \\ \Leftrightarrow \Pr[L] &= N \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]. \end{aligned}$$

- \mathcal{E} is p.p.t. $\Rightarrow N$ is polynomial
- DH assumption $\Rightarrow \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]$ is negligible

Hence, $\Pr[L]$ is negligible.

Security Proof: Bounding $\Pr[L]$

Event L : during attack, \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$.

We show that $\Pr[L]$ is negligible under the DH assumption.

Define p.p.t. algorithm \mathcal{E}' , using p.p.t. \mathcal{E} as a subroutine:

- ➊ take h_A and h_B as inputs;
- ➋ run \mathcal{E} on these inputs, while recording all \mathcal{H} queries made by \mathcal{E} ;
- ➌ return an \mathcal{H} query (made by \mathcal{E}) at random.

Let N denote total number of \mathcal{H} queries. Then:

$$\begin{aligned} \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}] &= \Pr[L]/N \\ \Leftrightarrow \Pr[L] &= N \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]. \end{aligned}$$

- \mathcal{E} is p.p.t. $\Rightarrow N$ is polynomial
- DH assumption $\Rightarrow \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]$ is negligible

Hence, $\Pr[L]$ is negligible.

Security Proof: Bounding $\Pr[L]$

Event L : during attack, \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$.

We show that $\Pr[L]$ is negligible under the DH assumption.

Define p.p.t. algorithm \mathcal{E}' , using p.p.t. \mathcal{E} as a subroutine:

- ➊ take h_A and h_B as inputs;
- ➋ run \mathcal{E} on these inputs, while recording all \mathcal{H} queries made by \mathcal{E} ;
- ➌ return an \mathcal{H} query (made by \mathcal{E}) at random.

Let N denote total number of \mathcal{H} queries. Then:

$$\begin{aligned} \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}] &= \Pr[L]/N \\ \Leftrightarrow \Pr[L] &= N \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]. \end{aligned}$$

- \mathcal{E} is p.p.t. $\Rightarrow N$ is polynomial
- DH assumption $\Rightarrow \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]$ is negligible

Hence, $\Pr[L]$ is negligible.

Security Proof: Bounding $\Pr[L]$

Event L : during attack, \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$.

We show that $\Pr[L]$ is negligible under the DH assumption.

Define p.p.t. algorithm \mathcal{E}' , using p.p.t. \mathcal{E} as a subroutine:

- ➊ take h_A and h_B as inputs;
- ➋ run \mathcal{E} on these inputs, while recording all \mathcal{H} queries made by \mathcal{E} ;
- ➌ return an \mathcal{H} query (made by \mathcal{E}) at random.

Let N denote total number of \mathcal{H} queries. Then:

$$\begin{aligned} \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}] &= \Pr[L]/N \\ \Leftrightarrow \Pr[L] &= N \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]. \end{aligned}$$

- \mathcal{E} is p.p.t. $\Rightarrow N$ is polynomial
- DH assumption $\Rightarrow \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]$ is negligible

Hence, $\Pr[L]$ is negligible.

Security Proof: Bounding $\Pr[L]$

Event L : during attack, \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$.

We show that $\Pr[L]$ is negligible under the DH assumption.

Define p.p.t. algorithm \mathcal{E}' , using p.p.t. \mathcal{E} as a subroutine:

- ➊ take h_A and h_B as inputs;
- ➋ run \mathcal{E} on these inputs, while recording all \mathcal{H} queries made by \mathcal{E} ;
- ➌ return an \mathcal{H} query (made by \mathcal{E}) at random.

Let N denote total number of \mathcal{H} queries. Then:

$$\begin{aligned} \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}] &= \Pr[L]/N \\ \Leftrightarrow \Pr[L] &= N \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]. \end{aligned}$$

- \mathcal{E} is p.p.t. $\Rightarrow N$ is polynomial
 - DH assumption $\Rightarrow \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]$ is negligible
- Hence, $\Pr[L]$ is negligible.



Security Proof: Bounding $\Pr[L]$

Event L : during attack, \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$.

We show that $\Pr[L]$ is negligible under the DH assumption.

Define p.p.t. algorithm \mathcal{E}' , using p.p.t. \mathcal{E} as a subroutine:

- ➊ take h_A and h_B as inputs;
- ➋ run \mathcal{E} on these inputs, while recording all \mathcal{H} queries made by \mathcal{E} ;
- ➌ return an \mathcal{H} query (made by \mathcal{E}) at random.

Let N denote total number of \mathcal{H} queries. Then:

$$\begin{aligned} \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}] &= \Pr[L]/N \\ \Leftrightarrow \Pr[L] &= N \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]. \end{aligned}$$

- \mathcal{E} is p.p.t. $\Rightarrow N$ is polynomial
- DH assumption $\Rightarrow \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]$ is negligible

Hence, $\Pr[L]$ is negligible.

Security Proof: Bounding $\Pr[L]$

Event L : during attack, \mathcal{E} queries \mathcal{H} on input $g^{x_A x_B}$.

We show that $\Pr[L]$ is negligible under the DH assumption.

Define p.p.t. algorithm \mathcal{E}' , using p.p.t. \mathcal{E} as a subroutine:

- ➊ take h_A and h_B as inputs;
- ➋ run \mathcal{E} on these inputs, while recording all \mathcal{H} queries made by \mathcal{E} ;
- ➌ return an \mathcal{H} query (made by \mathcal{E}) at random.

Let N denote total number of \mathcal{H} queries. Then:

$$\begin{aligned} \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}] &= \Pr[L]/N \\ \Leftrightarrow \Pr[L] &= N \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]. \end{aligned}$$

- \mathcal{E} is p.p.t. $\Rightarrow N$ is polynomial
- DH assumption $\Rightarrow \Pr[\mathcal{E}'(h_A, h_B) = g^{x_A x_B}]$ is negligible

Hence, $\Pr[L]$ is negligible.

Exercise

Exercise 2.5

Extend the above analysis to the case that the a priori probabilities are given by $\Pr[f(u) = 0] = p_0$ and $\Pr[f(u) = 1] = p_1 = 1 - p_0$.

ElGamal Encryption

ElGamal cryptosystem, for security parameter k

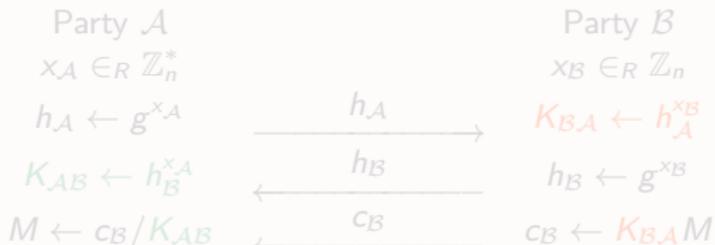
Key generation. Pick group $\langle g \rangle$ of order n at random among “size” k groups.
 Pick private key $x \in_R \mathbb{Z}_n^*$. Set public key as $h = g^x$.

Encryption. Given plaintext $M \in \langle g \rangle$ and public key h . The ciphertext is the pair $(g^u, h^u M)$, where $u \in_R \mathbb{Z}_n$.

Decryption. Given ciphertext (A, B) , the plaintext is recovered as $M = B/A^x$, using private key x .

Semantically secure under **DDH assumption**: ciphertext leaks no information.

ElGamal encryption \simeq Diffie-Hellman Key Exchange plus transfer of M



ElGamal Encryption

ElGamal cryptosystem, for security parameter k

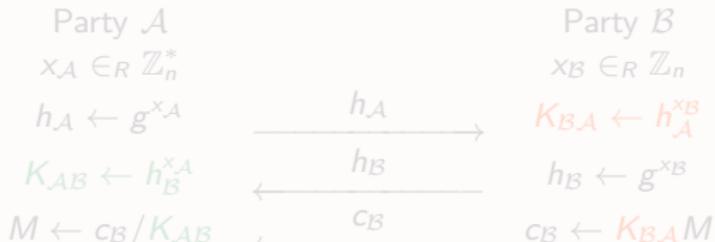
Key generation. Pick group $\langle g \rangle$ of order n at random among “size” k groups.
 Pick private key $x \in_R \mathbb{Z}_n^*$. Set public key as $h = g^x$.

Encryption. Given plaintext $M \in \langle g \rangle$ and public key h . The ciphertext is the pair $(g^u, h^u M)$, where $u \in_R \mathbb{Z}_n$.

Decryption. Given ciphertext (A, B) , the plaintext is recovered as $M = B/A^x$, using private key x .

Semantically secure under **DDH assumption**: ciphertext leaks no information.

ElGamal encryption \simeq Diffie-Hellman Key Exchange plus transfer of M



ElGamal Encryption

ElGamal cryptosystem, for security parameter k

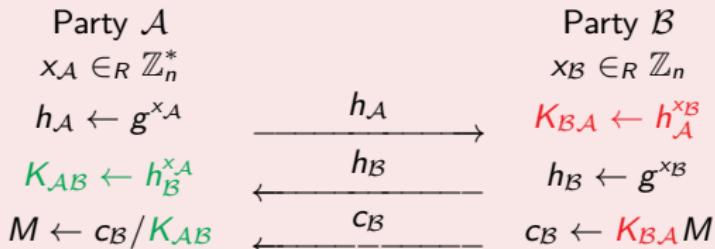
Key generation. Pick group $\langle g \rangle$ of order n at random among “size” k groups.
 Pick private key $x \in_R \mathbb{Z}_n^*$. Set public key as $h = g^x$.

Encryption. Given plaintext $M \in \langle g \rangle$ and public key h . The ciphertext is the pair $(g^u, h^u M)$, where $u \in_R \mathbb{Z}_n$.

Decryption. Given ciphertext (A, B) , the plaintext is recovered as
 $M = B/A^x$, using private key x .

Semantically secure under **DDH assumption**: ciphertext leaks no information.

ElGamal encryption \simeq Diffie-Hellman Key Exchange plus transfer of M



Exercise

Exercise 2.6

Show how to break the ElGamal cryptosystem for $\langle g \rangle = \mathbb{Z}_p^*$, with $p = 2p' + 1$, p, p' both prime. Focus on the case that $M \in \{1, g\}$, and show how to recover M .

Practical ElGamal Encryption

Cryptographic hash function $H : \langle g \rangle \rightarrow \{0, 1\}^k$.

Practical ElGamal cryptosystem, using H

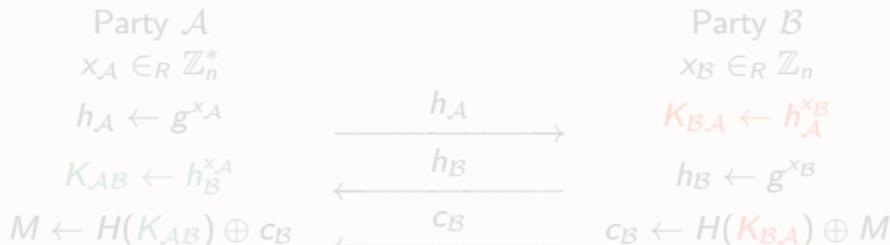
Key generation. As above.

Encryption. Given plaintext $M \in \{0, 1\}^k$ and public key h . The ciphertext is the pair $(g^u, H(h^u) \oplus M)$, where $u \in_R \mathbb{Z}_n$.

Decryption. Given ciphertext (A, B) , the plaintext is recovered as $M = H(A^x) \oplus B$, using private key x .

Semantically secure under **DH assumption**, in **Random Oracle model**.

Connection with Hashed DH Key Exchange



Practical ElGamal Encryption

Cryptographic hash function $H : \langle g \rangle \rightarrow \{0, 1\}^k$.

Practical ElGamal cryptosystem, using H

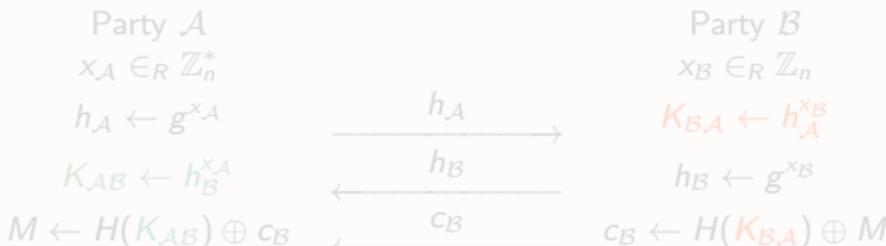
Key generation. As above.

Encryption. Given plaintext $M \in \{0, 1\}^k$ and public key h . The ciphertext is the pair $(g^u, H(h^u) \oplus M)$, where $u \in_R \mathbb{Z}_n$.

Decryption. Given ciphertext (A, B) , the plaintext is recovered as $M = H(A^x) \oplus B$, using private key x .

Semantically secure under **DH assumption**, in **Random Oracle model**.

Connection with Hashed DH Key Exchange



Practical ElGamal Encryption

Cryptographic hash function $H : \langle g \rangle \rightarrow \{0, 1\}^k$.

Practical ElGamal cryptosystem, using H

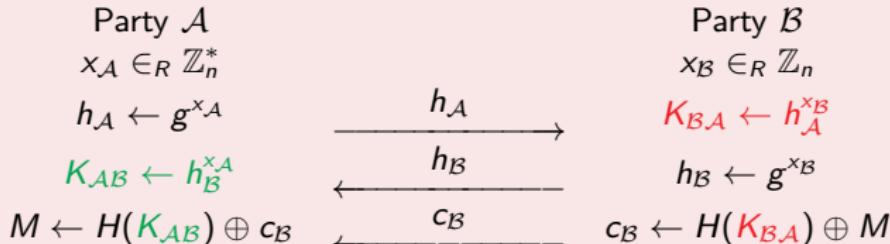
Key generation. As above.

Encryption. Given plaintext $M \in \{0, 1\}^k$ and public key h . The ciphertext is the pair $(g^u, H(h^u) \oplus M)$, where $u \in_R \mathbb{Z}_n$.

Decryption. Given ciphertext (A, B) , the plaintext is recovered as $M = H(A^x) \oplus B$, using private key x .

Semantically secure under **DH assumption**, in **Random Oracle model**.

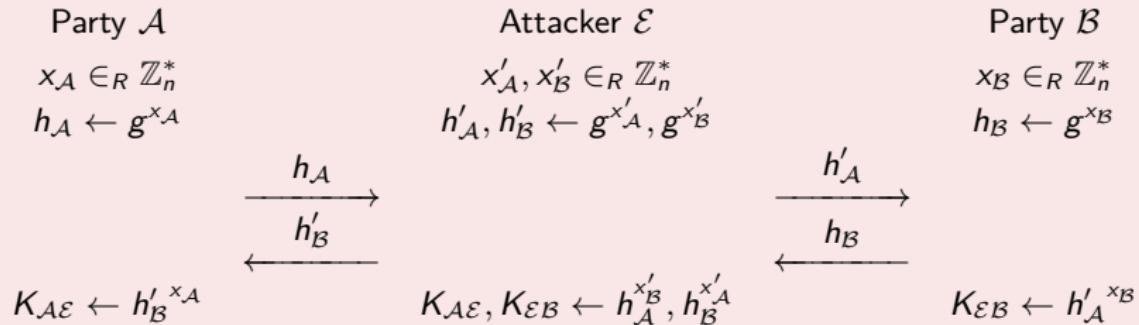
Connection with Hashed DH Key Exchange



Man-in-the-Middle Attacks

Active attacker may impersonate your protocol partner.

Classic man-in-the-middle attack

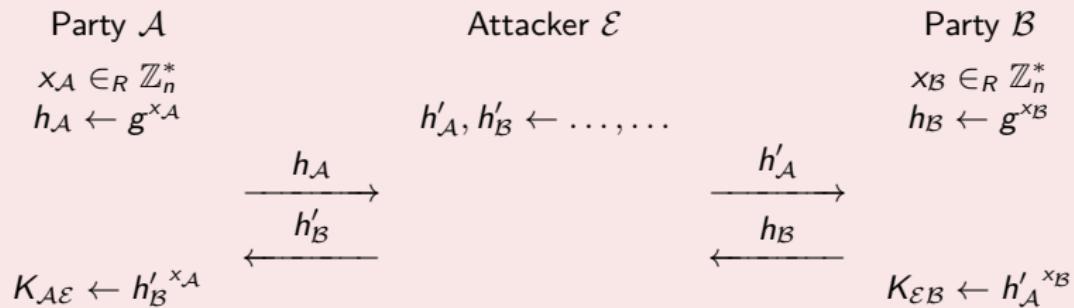


Man-in-the-middle \mathcal{E} :

- session key K_{AE} with \mathcal{A}
- session key K_{EB} with \mathcal{B}
- access to all traffic in clear between \mathcal{A} and \mathcal{B} ; can also modify things.

Man-in-the-Middle Attacks

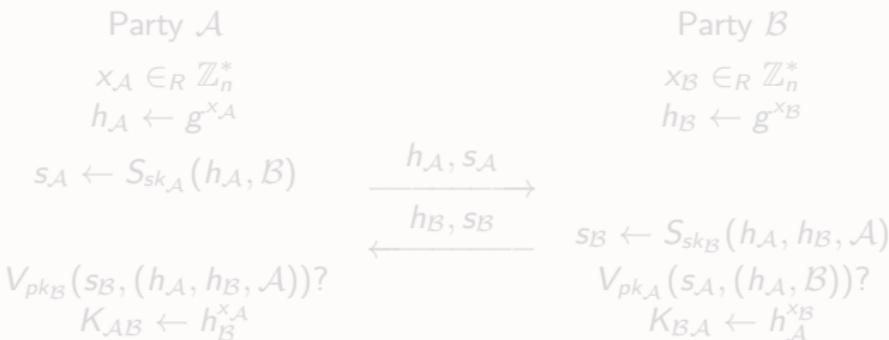
Arbitrary man-in-the-middle attacks



	h'_A	h'_B	K_{AE}	K_{EB}
Attack 0	1	1	1	1
Attack 1	g	g	g^{x_A}	g^{x_B}
Attack 2	$g^{x'_A}$	$g^{x'_B}$	$g^{x_A x'_B}$	$g^{x'_A x_B}$
Attack 3	$h_A g^u$	h_B	$g^{x_A x_B}$	$g^{(x_A + u) x_B}$
Attack 4	$1/h_A$	$1/h_B$	$g^{-x_A x_B}$	$g^{-x_A x_B}$

Signature-Based Authenticated Key Exchange

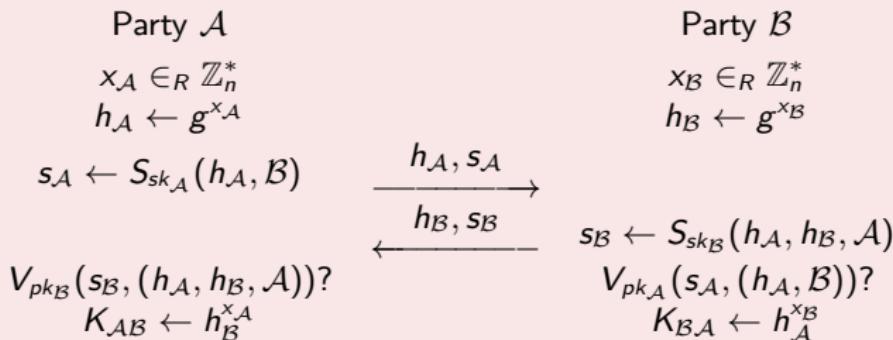
(pk_A, sk_A)	key pair of party \mathcal{A}
(pk_B, sk_B)	key pair of party \mathcal{B}
S	signature generation algorithm
V	signature verification algorithm



Secure under DDH assumption, assuming digital signature scheme is secure (against adaptive chosen-message attack).
 Use of digital signatures is somewhat costly.

Signature-Based Authenticated Key Exchange

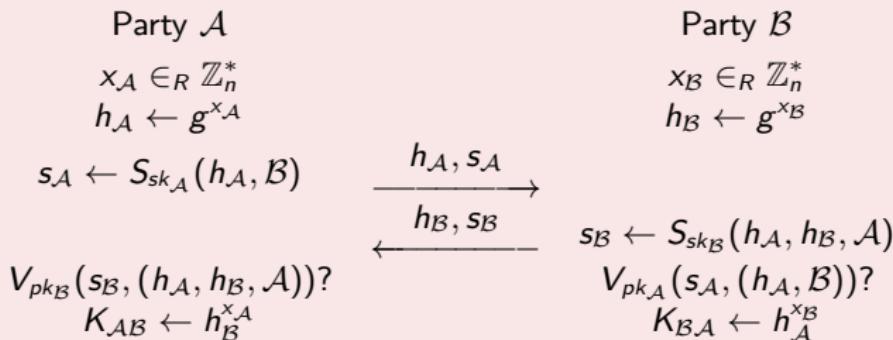
(pk_A, sk_A)	key pair of party \mathcal{A}
(pk_B, sk_B)	key pair of party \mathcal{B}
S	signature generation algorithm
V	signature verification algorithm



Secure under DDH assumption, assuming digital signature scheme is secure (against adaptive chosen-message attack).
 Use of digital signatures is somewhat costly.

Signature-Based Authenticated Key Exchange

(pk_A, sk_A)	key pair of party \mathcal{A}
(pk_B, sk_B)	key pair of party \mathcal{B}
S	signature generation algorithm
V	signature verification algorithm



Secure under DDH assumption, assuming digital signature scheme is secure (against adaptive chosen-message attack).
 Use of digital signatures is somewhat costly.

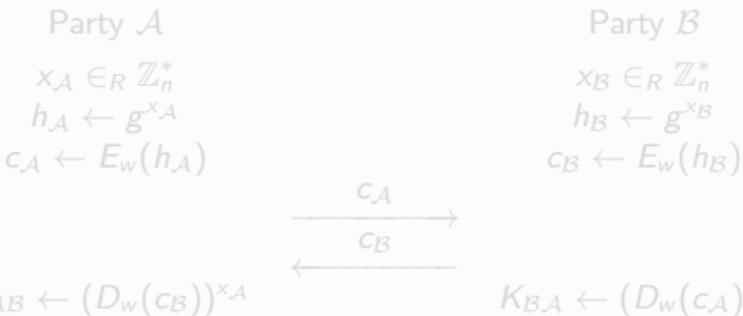
Password-Based Authenticated Key Exchange

$w \in \{0, 1\}^*$

$E, D : \{0, 1\}^* \times \langle g \rangle \rightarrow \langle g \rangle$

password known to \mathcal{A} and \mathcal{B} only

symmetric encryption/decryption algorithms



Then $K_{AB} = K_{BA} = g^{x_A x_B}$.

Passive attacker sees $E_w(h_A)$ and $E_w(h_B)$. **No information** on w , since h_A and h_B are random and unknown to the attacker. But, use of K in ensuing session may reveal information on K , hence on w .

Active attacker may try w' by sending $E_{w'}(h_A)$ on behalf of \mathcal{A} . When session succeeds, w' is correct. Best attacker can do is trying passwords one at a time. No off-line dictionary attack.

Password-Based Authenticated Key Exchange

$w \in \{0, 1\}^*$

$E, D : \{0, 1\}^* \times \langle g \rangle \rightarrow \langle g \rangle$

password known to \mathcal{A} and \mathcal{B} only

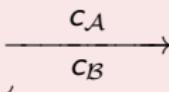
symmetric encryption/decryption algorithms

Party \mathcal{A}

$$\begin{aligned}x_{\mathcal{A}} &\in_R \mathbb{Z}_n^* \\ h_{\mathcal{A}} &\leftarrow g^{x_{\mathcal{A}}} \\ c_{\mathcal{A}} &\leftarrow E_w(h_{\mathcal{A}})\end{aligned}$$

Party \mathcal{B}

$$\begin{aligned}x_{\mathcal{B}} &\in_R \mathbb{Z}_n^* \\ h_{\mathcal{B}} &\leftarrow g^{x_{\mathcal{B}}} \\ c_{\mathcal{B}} &\leftarrow E_w(h_{\mathcal{B}})\end{aligned}$$



$K_{\mathcal{A}\mathcal{B}} \leftarrow (D_w(c_{\mathcal{B}}))^{x_{\mathcal{A}}}$

$K_{\mathcal{B}\mathcal{A}} \leftarrow (D_w(c_{\mathcal{A}}))^{x_{\mathcal{B}}}$

Then $K_{\mathcal{A}\mathcal{B}} = K_{\mathcal{B}\mathcal{A}} = g^{x_{\mathcal{A}}x_{\mathcal{B}}}$.

Passive attacker sees $E_w(h_{\mathcal{A}})$ and $E_w(h_{\mathcal{B}})$. **No information** on w , since $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$ are random and unknown to the attacker. But, use of K in ensuing session may reveal information on K , hence on w .

Active attacker may try w' by sending $E_{w'}(h_{\mathcal{A}})$ on behalf of \mathcal{A} . When session succeeds, w' is correct. Best attacker can do is trying passwords one at a time. No off-line dictionary attack.



Password-Based Authenticated Key Exchange

$w \in \{0, 1\}^*$

$E, D : \{0, 1\}^* \times \langle g \rangle \rightarrow \langle g \rangle$

password known to \mathcal{A} and \mathcal{B} only

symmetric encryption/decryption algorithms

Party \mathcal{A}

$x_{\mathcal{A}} \in_R \mathbb{Z}_n^*$

$h_{\mathcal{A}} \leftarrow g^{x_{\mathcal{A}}}$

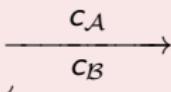
$c_{\mathcal{A}} \leftarrow E_w(h_{\mathcal{A}})$

Party \mathcal{B}

$x_{\mathcal{B}} \in_R \mathbb{Z}_n^*$

$h_{\mathcal{B}} \leftarrow g^{x_{\mathcal{B}}}$

$c_{\mathcal{B}} \leftarrow E_w(h_{\mathcal{B}})$



$K_{AB} \leftarrow (D_w(c_B))^{x_{\mathcal{A}}}$

$K_{BA} \leftarrow (D_w(c_A))^{x_{\mathcal{B}}}$

Then $K_{AB} = K_{BA} = g^{x_{\mathcal{A}} x_{\mathcal{B}}}$.

Passive attacker sees $E_w(h_{\mathcal{A}})$ and $E_w(h_{\mathcal{B}})$. **No information** on w , since $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$ are random and unknown to the attacker. But, use of K in ensuing session may reveal information on K , hence on w .

Active attacker may try w' by sending $E_{w'}(h_{\mathcal{A}})$ on behalf of \mathcal{A} . When session succeeds, w' is correct. Best attacker can do is trying passwords one at a time. No off-line dictionary attack.

Password-Based Authenticated Key Exchange

$w \in \{0, 1\}^*$

$E, D : \{0, 1\}^* \times \langle g \rangle \rightarrow \langle g \rangle$

password known to \mathcal{A} and \mathcal{B} only

symmetric encryption/decryption algorithms

Party \mathcal{A}

$x_{\mathcal{A}} \in_R \mathbb{Z}_n^*$

$h_{\mathcal{A}} \leftarrow g^{x_{\mathcal{A}}}$

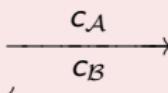
$c_{\mathcal{A}} \leftarrow E_w(h_{\mathcal{A}})$

Party \mathcal{B}

$x_{\mathcal{B}} \in_R \mathbb{Z}_n^*$

$h_{\mathcal{B}} \leftarrow g^{x_{\mathcal{B}}}$

$c_{\mathcal{B}} \leftarrow E_w(h_{\mathcal{B}})$



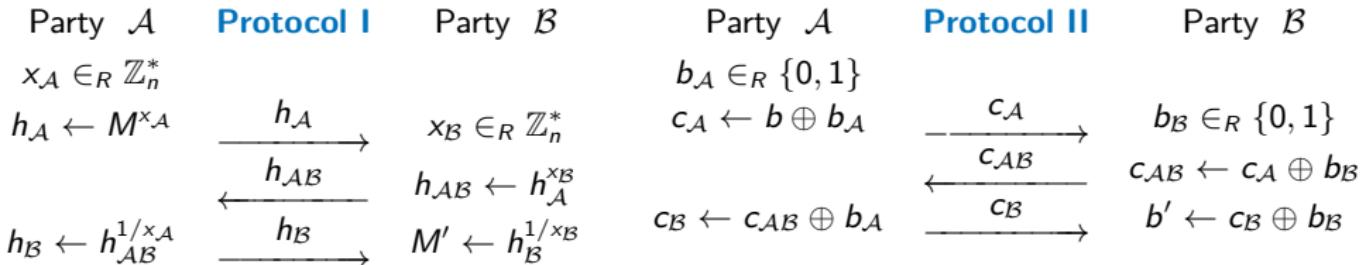
$K_{AB} \leftarrow (D_w(c_B))^{x_{\mathcal{A}}}$

$K_{BA} \leftarrow (D_w(c_A))^{x_{\mathcal{B}}}$

$\text{Then } K_{AB} = K_{BA} = g^{x_{\mathcal{A}} x_{\mathcal{B}}}.$

Passive attacker sees $E_w(h_{\mathcal{A}})$ and $E_w(h_{\mathcal{B}})$. **No information** on w , since $h_{\mathcal{A}}$ and $h_{\mathcal{B}}$ are random and unknown to the attacker. But, use of K in ensuing session may reveal information on K , hence on w .

Active attacker may try w' by sending $E_{w'}(h_{\mathcal{A}})$ on behalf of \mathcal{A} . When session succeeds, w' is correct. Best attacker can do is trying passwords one at a time. No off-line dictionary attack.



Exercise 2.7

Consider Protocols I and II for sending a plaintext securely from \mathcal{A} to \mathcal{B} over an insecure channel. In Protocol I plaintext $M \in \langle g \rangle$, $M \neq 1$; in Protocol II plaintext $b \in \{0, 1\}$. The object of both protocols is that the plaintext remains completely hidden from other parties than \mathcal{A} and \mathcal{B} , and that the plaintext cannot be modified by other parties than \mathcal{A} or \mathcal{B} .

First, verify that $M' = M$ and $b' = b$ if \mathcal{A} and \mathcal{B} follow protocols I and II, respectively. Next, determine for protocol I whether it is secure against passive attacks, and whether it is secure against active attacks. If secure, describe the relevant computational assumption (if any); if insecure, show an attack. Do the same for protocol II.

Commitments

Party \mathcal{A}

Party \mathcal{B}

Commit phase :



Put message in
commitment.
Keep key.

Commitment
stored for later.

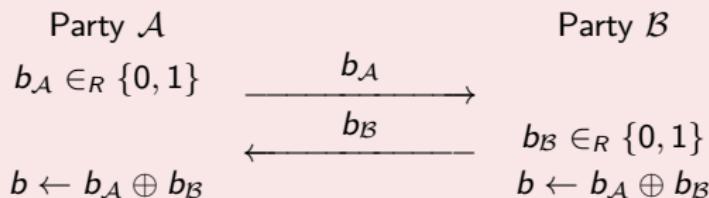
Reveal phase :



Check message
in commitment.

Coin Flipping by Telephone

Mutually random bit?



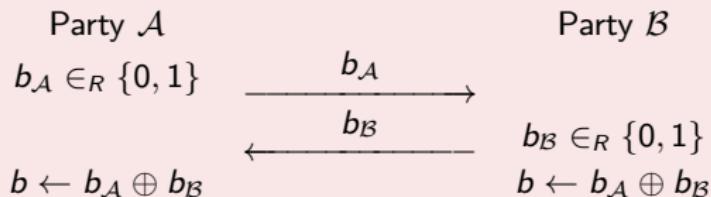
Protocol flaw:

Party B can set $b_B = b_A \oplus b^*$ for bit value b^* of its liking.

Insider attack! By your protocol partner.

Coin Flipping by Telephone

Mutually random bit?



Protocol flaw:

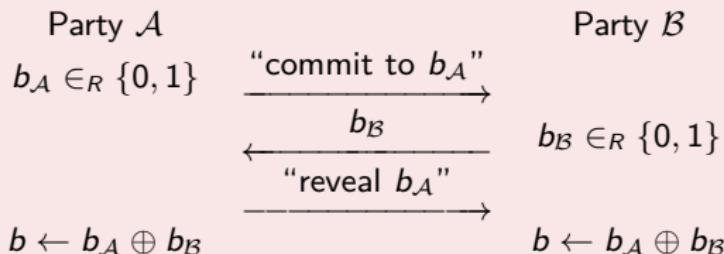
Party B can set $b_B = b_A \oplus b^*$ for bit value b^* of its liking.

Insider attack! By your protocol partner.

Coin Flipping by Telephone

Use commitments.

Mutually random bit!



If \mathcal{A} or \mathcal{B} is honest, bit b is distributed uniformly at random, provided:

- commitment **hides** b_A from party \mathcal{B}
- commitment **binds** party \mathcal{A} to b_A (no change possible).

Asymmetric functionality.

Definition 3.1 (Commitment schemes)

Let $\text{commit} : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a deterministic polynomial time algorithm. Non-interactive **commitment scheme** between sender and receiver:

Commit Phase. Protocol in which sender commits to $x \in \{0, 1\}^*$ by computing $C = \text{commit}(u, x)$ with $u \in_R \{0, 1\}^k$, and sending C to receiver. Receiver stores C for later use.

Reveal Phase. Protocol in which sender opens commitment $C = \text{commit}(u, x)$ by sending u and x to receiver. Receiver computes $\text{commit}(u, x)$ and verifies equality to C .

Security for sender: $\text{commit}(u, x)$ should not leak information on x .

Let $X = \{\text{commit}(u, 0) : u \in_R \{0, 1\}^k\}$, $Y = \{\text{commit}(u, 1) : u \in_R \{0, 1\}^k\}$.

Computationally hiding. X and Y **computationally** indistinguishable.

Information-theoretically hiding. X and Y **statistically** indistinguishable.

Security for receiver: $\text{commit}(u, x)$ can be opened in one way only.

Computationally binding. Adversary restricted to **p.p.t. algorithm**.

Information-theoretically binding. Adversary **unlimitedly powerful**.

Asymmetric functionality.

Definition 3.1 (Commitment schemes)

Let $\text{commit} : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a deterministic polynomial time algorithm. Non-interactive **commitment scheme** between sender and receiver:

Commit Phase. Protocol in which sender commits to $x \in \{0, 1\}^*$ by computing $C = \text{commit}(u, x)$ with $u \in_R \{0, 1\}^k$, and sending C to receiver. Receiver stores C for later use.

Reveal Phase. Protocol in which sender opens commitment $C = \text{commit}(u, x)$ by sending u and x to receiver. Receiver computes $\text{commit}(u, x)$ and verifies equality to C .

Security for sender: $\text{commit}(u, x)$ should not leak information on x .

Let $X = \{\text{commit}(u, 0) : u \in_R \{0, 1\}^k\}$, $Y = \{\text{commit}(u, 1) : u \in_R \{0, 1\}^k\}$.

Computationally **hiding**. X and Y **computationally** indistinguishable.

Information-theoretically **hiding**. X and Y **statistically** indistinguishable.

Security for receiver: $\text{commit}(u, x)$ can be opened in one way only.

Computationally **binding**. Adversary restricted to **p.p.t. algorithm**.

Information-theoretically **binding**. Adversary **unlimitedly powerful**.

Asymmetric functionality.

Definition 3.1 (Commitment schemes)

Let $\text{commit} : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a deterministic polynomial time algorithm. Non-interactive **commitment scheme** between sender and receiver:

Commit Phase. Protocol in which sender commits to $x \in \{0, 1\}^*$ by computing $C = \text{commit}(u, x)$ with $u \in_R \{0, 1\}^k$, and sending C to receiver. Receiver stores C for later use.

Reveal Phase. Protocol in which sender opens commitment $C = \text{commit}(u, x)$ by sending u and x to receiver. Receiver computes $\text{commit}(u, x)$ and verifies equality to C .

Security for sender: $\text{commit}(u, x)$ should not leak information on x .

Let $X = \{\text{commit}(u, 0) : u \in_R \{0, 1\}^k\}$, $Y = \{\text{commit}(u, 1) : u \in_R \{0, 1\}^k\}$.

Computationally hiding. X and Y **computationally** indistinguishable.

Information-theoretically hiding. X and Y **statistically** indistinguishable.

Security for receiver: $\text{commit}(u, x)$ can be opened in one way only.

Computationally binding. Adversary restricted to **p.p.t. algorithm**.

Information-theoretically binding. Adversary **unlimitedly powerful**.

Given cryptographic hash function H .

Hash-based bit commitment scheme

$$\text{commit}_0(u, x) = H(u, x), \quad u \in_R \{0, 1\}^k$$

Computationally binding:

- Implied by collision resistance:
 $H(u, x) = H(u', 1 - x)$ is a collision.
- Hence, also binding in the random oracle model.

Computationally hiding:

- Preimage resistance necessary **but not sufficient**:
 $H(u, x)$ might leak partial information on input u, x
(See also Exercise 1.39 on *partial preimage resistance*.)
- Hiding in the random oracle model:
 $H(u, x)$ is uniformly random and independent of x

Given cryptographic hash function H .

Hash-based bit commitment scheme

$$\text{commit}_0(u, x) = H(u, x), \quad u \in_R \{0, 1\}^k$$

Computationally binding:

- Implied by collision resistance:
 $H(u, x) = H(u', 1 - x)$ is a collision.
- Hence, also binding in the random oracle model.

Computationally hiding:

- Preimage resistance necessary **but not sufficient**:
 $H(u, x)$ might leak partial information on input u, x
(See also Exercise 1.39 on *partial preimage resistance*.)
- Hiding in the random oracle model:
 $H(u, x)$ is uniformly random and independent of x

Given cryptographic hash function H .

Hash-based bit commitment scheme

$$\text{commit}_0(u, x) = H(u, x), \quad u \in_R \{0, 1\}^k$$

Computationally binding:

- Implied by collision resistance:
 $H(u, x) = H(u', 1 - x)$ is a collision.
- Hence, also binding in the random oracle model.

Computationally hiding:

- Preimage resistance necessary **but not sufficient**:
 $H(u, x)$ might leak partial information on input u, x
(See also Exercise 1.39 on *partial preimage resistance*.)
- Hiding in the random oracle model:
 $H(u, x)$ is uniformly random and independent of x

3.1 Definition

3.2 Examples

3.3 Homomorphic Commitments

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

Two Complementary Schemes

DL setting: generators g, h such that nobody knows $\log_g h$.

Pedersen's bit commitment scheme

$$\text{commit}_1(u, x) = g^u h^x, \quad u \in_R \mathbb{Z}_n$$

Computationally binding under DL assumption:

$$\text{commit}_1(u, x) = \text{commit}_1(u', 1-x) \Leftrightarrow g^u h^x = g^{u'} h^{1-x} \Leftrightarrow \log_g h = \frac{u - u'}{1 - 2x}$$

Information-theoretically hiding: $g^u h^x$ is statistically independent of x .

EIGamal-based bit commitment scheme

$$\text{commit}_2(u, x) = (g^u, h^{u+x}), \quad u \in_R \mathbb{Z}_n$$

Information-theoretically binding:

$$\text{commit}_2(u, x) = \text{commit}_2(u', 1-x) \Leftrightarrow (g^u, h^{u+x}) = (g^{u'}, h^{u'+1-x}) \Leftrightarrow \text{false}$$

Computationally hiding under DDH assumption (DL assumption not sufficient).



3.1 Definition

3.2 Examples

3.3 Homomorphic Commitments

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

Two Complementary Schemes

DL setting: generators g, h such that nobody knows $\log_g h$.

Pedersen's bit commitment scheme

$$\text{commit}_1(u, x) = g^u h^x, \quad u \in_R \mathbb{Z}_n$$

Computationally binding under DL assumption:

$$\text{commit}_1(u, x) = \text{commit}_1(u', 1-x) \Leftrightarrow g^u h^x = g^{u'} h^{1-x} \Leftrightarrow \log_g h = \frac{u - u'}{1 - 2x}$$

Information-theoretically hiding: $g^u h^x$ is statistically independent of x .

EIGamal-based bit commitment scheme

$$\text{commit}_2(u, x) = (g^u, h^{u+x}), \quad u \in_R \mathbb{Z}_n$$

Information-theoretically binding:

$$\text{commit}_2(u, x) = \text{commit}_2(u', 1-x) \Leftrightarrow (g^u, h^{u+x}) = (g^{u'}, h^{u'+1-x}) \Leftrightarrow \text{false}$$

Computationally hiding under DDH assumption (DL assumption not sufficient).



3.1 Definition

3.2 Examples

3.3 Homomorphic Commitments

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

Two Complementary Schemes

DL setting: generators g, h such that nobody knows $\log_g h$.

Pedersen's bit commitment scheme

$$\text{commit}_1(u, x) = g^u h^x, \quad u \in_R \mathbb{Z}_n$$

Computationally binding under DL assumption:

$$\text{commit}_1(u, x) = \text{commit}_1(u', 1 - x) \Leftrightarrow g^u h^x = g^{u'} h^{1-x} \Leftrightarrow \log_g h = \frac{u - u'}{1 - 2x}$$

Information-theoretically hiding: $g^u h^x$ is statistically independent of x .

EIGamal-based bit commitment scheme

$$\text{commit}_2(u, x) = (g^u, h^{u+x}), \quad u \in_R \mathbb{Z}_n$$

Information-theoretically binding:

$$\text{commit}_2(u, x) = \text{commit}_2(u', 1 - x) \Leftrightarrow (g^u, h^{u+x}) = (g^{u'}, h^{u'+1-x}) \Leftrightarrow \text{false}$$

Computationally hiding under DDH assumption (DL assumption not sufficient).



3.1 Definition

3.2 Examples

3.3 Homomorphic Commitments

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

Two Complementary Schemes

DL setting: generators g, h such that nobody knows $\log_g h$.

Pedersen's bit commitment scheme

$$\text{commit}_1(u, x) = g^u h^x, \quad u \in_R \mathbb{Z}_n$$

Computationally binding under DL assumption:

$$\text{commit}_1(u, x) = \text{commit}_1(u', 1 - x) \Leftrightarrow g^u h^x = g^{u'} h^{1-x} \Leftrightarrow \log_g h = \frac{u - u'}{1 - 2x}$$

Information-theoretically hiding: $g^u h^x$ is statistically independent of x .

EIGamal-based bit commitment scheme

$$\text{commit}_2(u, x) = (g^u, h^{u+x}), \quad u \in_R \mathbb{Z}_n$$

Information-theoretically binding:

$$\text{commit}_2(u, x) = \text{commit}_2(u', 1 - x) \Leftrightarrow (g^u, h^{u+x}) = (g^{u'}, h^{u'+1-x}) \Leftrightarrow \text{false}$$

Computationally hiding under DDH assumption (DL assumption not sufficient).



3.1 Definition

3.2 Examples

3.3 Homomorphic Commitments

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

Two Complementary Schemes

DL setting: generators g, h such that nobody knows $\log_g h$.

Pedersen's bit commitment scheme

$$\text{commit}_1(u, x) = g^u h^x, \quad u \in_R \mathbb{Z}_n$$

Computationally binding under DL assumption:

$$\text{commit}_1(u, x) = \text{commit}_1(u', 1 - x) \Leftrightarrow g^u h^x = g^{u'} h^{1-x} \Leftrightarrow \log_g h = \frac{u - u'}{1 - 2x}$$

Information-theoretically hiding: $g^u h^x$ is statistically independent of x .

EIGamal-based bit commitment scheme

$$\text{commit}_2(u, x) = (g^u, h^{u+x}), \quad u \in_R \mathbb{Z}_n$$

Information-theoretically binding:

$$\text{commit}_2(u, x) = \text{commit}_2(u', 1 - x) \Leftrightarrow (g^u, h^{u+x}) = (g^{u'}, h^{u'+1-x}) \Leftrightarrow \text{false}$$

Computationally hiding under DDH assumption (DL assumption not sufficient).



3.1 Definition

3.2 Examples

3.3 Homomorphic Commitments

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

Two Complementary Schemes

DL setting: generators g, h such that nobody knows $\log_g h$.

Pedersen's bit commitment scheme

$$\text{commit}_1(u, x) = g^u h^x, \quad u \in_R \mathbb{Z}_n$$

Computationally binding under DL assumption:

$$\text{commit}_1(u, x) = \text{commit}_1(u', 1 - x) \Leftrightarrow g^u h^x = g^{u'} h^{1-x} \Leftrightarrow \log_g h = \frac{u - u'}{1 - 2x}$$

Information-theoretically hiding: $g^u h^x$ is statistically independent of x .

EIGamal-based bit commitment scheme

$$\text{commit}_2(u, x) = (g^u, h^{u+x}), \quad u \in_R \mathbb{Z}_n$$

Information-theoretically binding:

$$\text{commit}_2(u, x) = \text{commit}_2(u', 1 - x) \Leftrightarrow (g^u, h^{u+x}) = (g^{u'}, h^{u'+1-x}) \Leftrightarrow \text{false}$$

Computationally hiding under DDH assumption (DL assumption not sufficient).



3.1 Definition

3.2 Examples

3.3 Homomorphic Commitments

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

Two Complementary Schemes

DL setting: generators g, h such that nobody knows $\log_g h$.

Pedersen's bit commitment scheme

$$\text{commit}_1(u, x) = g^u h^x, \quad u \in_R \mathbb{Z}_n$$

Computationally binding under DL assumption:

$$\text{commit}_1(u, x) = \text{commit}_1(u', 1 - x) \Leftrightarrow g^u h^x = g^{u'} h^{1-x} \Leftrightarrow \log_g h = \frac{u - u'}{1 - 2x}$$

Information-theoretically hiding: $g^u h^x$ is statistically independent of x .

EIGamal-based bit commitment scheme

$$\text{commit}_2(u, x) = (g^u, h^{u+x}), \quad u \in_R \mathbb{Z}_n$$

Information-theoretically binding:

$$\text{commit}_2(u, x) = \text{commit}_2(u', 1 - x) \Leftrightarrow (g^u, h^{u+x}) = (g^{u'}, h^{u'+1-x}) \Leftrightarrow \text{false}$$

Computationally hiding under DDH assumption (DL assumption not sufficient).



3.1 Definition

3.2 Examples

3.3 Homomorphic Commitments

3.2.1 Using a Cryptographic Hash Function

3.2.2 Using a Discrete Log Setting

3.2.3 Impossibility Result

Exercises

Exercise 3.2

Analyze the security properties of commit_1 and commit_2 for the case that $x \in \mathbb{Z}_n$.

Exercise 3.3

What happens if the receiver knows $\log_g h$ in scheme commit_1 ? Same question for scheme commit_2 .

Exercise 3.4

Discuss the security of the following commitment scheme for values $x \in \langle g \rangle$:

$$\text{commit}(u, x) = g^u x,$$

where $u \in_R \mathbb{Z}_n$. Is it binding? Is it hiding?



	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no u, u' exist** s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no u' exists** s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not information-theoretically hiding**.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no u, u' exist** s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no u' exists** s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not information-theoretically hiding**.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no u, u' exist** s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no u' exists** s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not information-theoretically hiding**.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no u, u' exist** s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no u' exists** s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not information-theoretically hiding**.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is information-theoretically binding.

Then no u, u' exist s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an unlimitedly powerful sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an unlimitedly powerful receiver notices that no u' exists s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is not information-theoretically hiding.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no** u, u' exist s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no** u' exists s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not** **information-theoretically hiding**.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no u, u' exist** s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no u' exists** s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not information-theoretically hiding**.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no u, u' exist** s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no u' exists** s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not information-theoretically hiding**.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no u, u' exist** s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no u' exists** s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not information-theoretically hiding**.

	binding	hiding
$\text{commit}_0(u, x) = H(u, x)$	computational	computational
$\text{commit}_1(u, x) = g^u h^x$	computational	information-theoretic
$\text{commit}_2(u, x) = (g^u, h^{u+x})$	information-theoretic	computational
$\text{commit}_3(u, x) = \text{impossible}$	information-theoretic	information-theoretic

Consider any bit commitment scheme commit_3 .

Assume commit_3 is **information-theoretically binding**.

Then **no u, u' exist** s.t. $\text{commit}_3(u, 0) = \text{commit}_3(u', 1)$, otherwise an **unlimitedly powerful** sender would find u, u' (by exhausting the finite set of possibilities).

But, if the sender commits to 0, say, using $C = \text{commit}_3(u, 0)$ for some u , an **unlimitedly powerful** receiver notices that **no u' exists** s.t. $C = \text{commit}_3(u', 1)$ (by exhausting the finite set of possibilities).

Hence the receiver knows that the committed value must be 0.

So, commit_3 is **not information-theoretically hiding**.

Homomorphic Commitments

Recall: homomorphic mapping preserves algebraic structure.

Example (Pedersen's commitment scheme)

$\text{commit}_1(u, x) = g^u h^x$, $u \in_R \mathbb{Z}_n$ and $x \in \mathbb{Z}_n$ is **additively** homomorphic:

$$\text{commit}_1(u, x) \text{commit}_1(u', x') = \text{commit}_1(u + u', x + x').$$

Multiplication on the left-hand side is in $\langle g \rangle$.

Additions on the right-hand side are in \mathbb{Z}_n ; $x + x'$ is **sum** of committed values.

Exercise 3.5 (See Exercise 1.35)

Quadratic Residuosity (QR) assumption states that QR problem is hard.

Let $y \in J_m$ denote a quadratic non-residue modulo m . Consider the following bit commitment scheme:

$$\text{commit}(u, x) = y^x u^2 \bmod m,$$

where $u \in_R \mathbb{Z}_m^$ and $x \in \{0, 1\}$. In what sense is the scheme binding? In what sense is the scheme hiding? In what sense is the scheme homomorphic?*



Homomorphic Commitments

Recall: homomorphic mapping preserves algebraic structure.

Example (Pedersen's commitment scheme)

$\text{commit}_1(u, x) = g^u h^x$, $u \in_R \mathbb{Z}_n$ and $x \in \mathbb{Z}_n$ is **additively** homomorphic:

$$\text{commit}_1(u, x) \text{commit}_1(u', x') = \text{commit}_1(u + u', x + x').$$

Multiplication on the left-hand side is in $\langle g \rangle$.

Additions on the right-hand side are in \mathbb{Z}_n ; $x + x'$ is **sum** of committed values.

Exercise 3.5 (See Exercise 1.35)

Quadratic Residuosity (QR) assumption states that QR problem is hard.

Let $y \in J_m$ denote a quadratic non-residue modulo m . Consider the following bit commitment scheme:

$$\text{commit}(u, x) = y^x u^2 \bmod m,$$

where $u \in_R \mathbb{Z}_m^$ and $x \in \{0, 1\}$. In what sense is the scheme binding? In what sense is the scheme hiding? In what sense is the scheme homomorphic?*

Homomorphic Commitments

Recall: homomorphic mapping preserves algebraic structure.

Example (Pedersen's commitment scheme)

$\text{commit}_1(u, x) = g^u h^x$, $u \in_R \mathbb{Z}_n$ and $x \in \mathbb{Z}_n$ is **additively** homomorphic:

$$\text{commit}_1(u, x) \text{commit}_1(u', x') = \text{commit}_1(u + u', x + x').$$

Multiplication on the left-hand side is in $\langle g \rangle$.

Additions on the right-hand side are in \mathbb{Z}_n ; $x + x'$ is **sum** of committed values.

Exercise 3.5 (See Exercise 1.35)

Quadratic Residuosity (QR) assumption states that QR problem is hard.

Let $y \in J_m$ denote a quadratic non-residue modulo m . Consider the following bit commitment scheme:

$$\text{commit}(u, x) = y^x u^2 \bmod m,$$

where $u \in_R \mathbb{Z}_m^$ and $x \in \{0, 1\}$. In what sense is the scheme binding? In what sense is the scheme hiding? In what sense is the scheme homomorphic?*



Identification protocol: two-party protocol to let **verifier** \mathcal{V} get convinced that **prover** \mathcal{P} is as claimed.

Examples: secure login, secured room.

Identification protocols may be based on (combinations of):

- *What you are.* Biometrics, such as fingerprints, iris scans, etc.
- *What you have.* Smart cards, SIM cards, or similar hardware tokens.
- *What you know.* Passwords, PIN codes, secret keys.

Our focus: purely **cryptographic** identification protocols, using **secret keys**.

Identification \neq message authentication
Identification \neq digital signatures
Identification \neq authenticated key exchange

Identification protocol: two-party protocol to let **verifier** \mathcal{V} get convinced that **prover** \mathcal{P} is as claimed.

Examples: secure login, secured room.

Identification protocols may be based on (combinations of):

- *What you are.* Biometrics, such as fingerprints, iris scans, etc.
- *What you have.* Smart cards, SIM cards, or similar hardware tokens.
- *What you know.* Passwords, PIN codes, secret keys.

Our focus: purely **cryptographic** identification protocols, using **secret keys**.

Identification \neq message authentication

Identification \neq digital signatures

Identification \neq authenticated key exchange

Identification protocol: two-party protocol to let **verifier** \mathcal{V} get convinced that **prover** \mathcal{P} is as claimed.

Examples: secure login, secured room.

Identification protocols may be based on (combinations of):

- *What you are.* Biometrics, such as fingerprints, iris scans, etc.
- *What you have.* Smart cards, SIM cards, or similar hardware tokens.
- *What you know.* Passwords, PIN codes, secret keys.

Our focus: purely **cryptographic** identification protocols, **using secret keys**.

Identification \neq message authentication

Identification \neq digital signatures

Identification \neq authenticated key exchange

Identification protocol: two-party protocol to let **verifier** \mathcal{V} get convinced that **prover** \mathcal{P} is as claimed.

Examples: secure login, secured room.

Identification protocols may be based on (combinations of):

- *What you are.* Biometrics, such as fingerprints, iris scans, etc.
- *What you have.* Smart cards, SIM cards, or similar hardware tokens.
- *What you know.* Passwords, PIN codes, secret keys.

Our focus: purely **cryptographic** identification protocols, **using secret keys**.

Identification \neq message authentication

Identification \neq digital signatures

Identification \neq authenticated key exchange

Identification Scheme

Registration protocol: once to set up a shared secret key or a public key pair.

Identification protocol: many runs, should withstand **impersonation attacks**

Passive attacks: eavesdropping, key only

Active attacks: guessing, cheating verifier, man-in-the-middle

Cheating verifier impersonation attack

- ① Attacker runs protocol (many times) as **cheating \mathcal{V}^*** with honest \mathcal{P} ;
- ② attacker runs protocol as cheating \mathcal{P}^* with honest \mathcal{V} .

Identification Scheme

Registration protocol: once to set up a shared secret key or a public key pair.

Identification protocol: many runs, should withstand **impersonation attacks**

Passive attacks: eavesdropping, key only

Active attacks: guessing, cheating verifier, man-in-the-middle

Cheating verifier impersonation attack

- ① Attacker runs protocol (many times) as **cheating \mathcal{V}^*** with honest \mathcal{P} ;
- ② attacker runs protocol as cheating \mathcal{P}^* with honest \mathcal{V} .

Identification Scheme

Registration protocol: once to set up a shared secret key or a public key pair.

Identification protocol: many runs, should withstand **impersonation attacks**

Passive attacks: eavesdropping, key only

Active attacks: guessing, cheating verifier, man-in-the-middle

Cheating verifier impersonation attack

- ① Attacker runs protocol (many times) as **cheating \mathcal{V}^*** with honest \mathcal{P} ;
- ② attacker runs protocol as cheating \mathcal{P}^* with honest \mathcal{V} .

Identification Scheme

Registration protocol: once to set up a shared secret key or a public key pair.

Identification protocol: many runs, should withstand **impersonation attacks**

Passive attacks: eavesdropping, key only

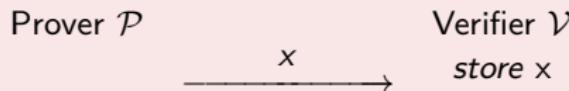
Active attacks: guessing, cheating verifier, man-in-the-middle

Cheating verifier impersonation attack

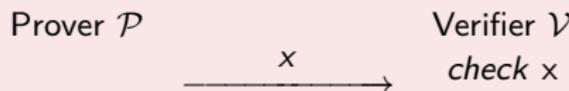
- ① Attacker runs protocol (many times) as **cheating \mathcal{V}^*** with honest \mathcal{P} ;
- ② attacker runs protocol as cheating \mathcal{P}^* with honest \mathcal{V} .

Password: usually a human-memorable string x .

Registration (over secure channel)



Identification



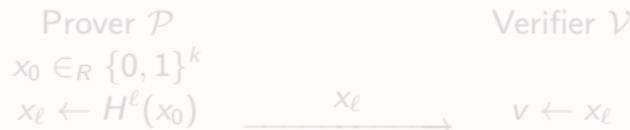
No security against eavesdropping attacks.

Lamport's Identification Scheme

Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

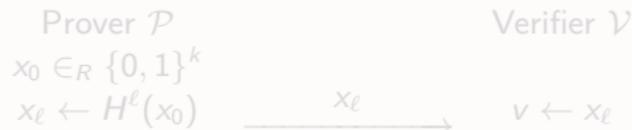
- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

Lamport's Identification Scheme

Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

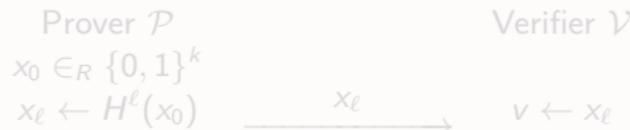
- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

Lamport's Identification Scheme

Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

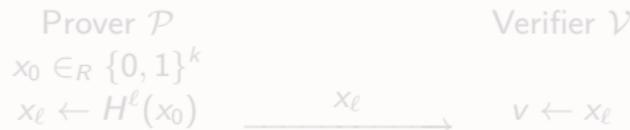
- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

Lamport's Identification Scheme

Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

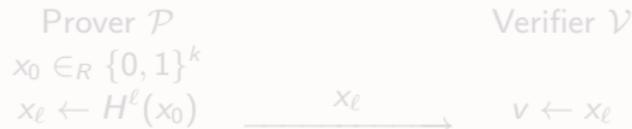
- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

Lamport's Identification Scheme

Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

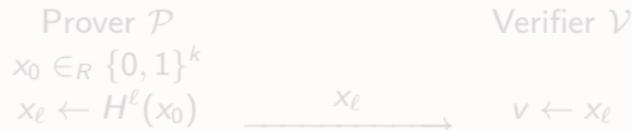
- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

Lamport's Identification Scheme

Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

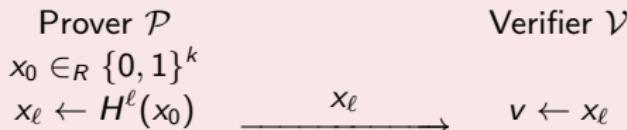
- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

Lamport's Identification Scheme

Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

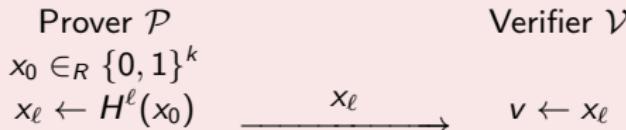


Lamport's Identification Scheme

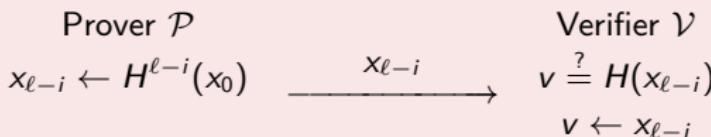
Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

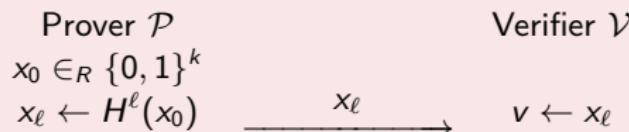


Lamport's Identification Scheme

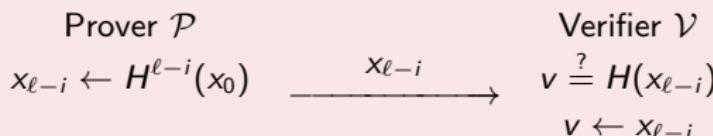
Hash chain with $x_{i+1} = H(x_i)$ for cryptographic hash function H :

$$x_0 \xrightarrow{H} x_1 \xrightarrow{H} x_2 \xrightarrow{H} \dots \xrightarrow{H} x_{\ell-2} \xrightarrow{H} x_{\ell-1} \xrightarrow{H} x_\ell$$

Registration (over secure channel)



Identification, i th run ($i = 1, \dots, \ell$)



Value v stored by verifier is not secret (but should be authentic).

- **Key-only attack:** in random oracle model.
- **Eavesdropping attack:** infeasible as $x_{\ell-i}$ does not help attacker in succeeding in subsequent runs of identification protocol.

Ultra Long Hash Chains

Hash chain of length ℓ allows for ℓ runs of identification protocol.

Suppose smart card must support maximum of $\ell = 2^{32}$ runs.

Direct computation of $H^{\ell-i}(x_0)$ impedes fast identification.

Remark 4.1 (Pebbling algorithms)

Prover	space	time
storing only x_0	$O(1)$	$O(\ell)$
storing all of $x_0, \dots, x_{\ell-i}$	$O(\ell)$	$O(1)$
pebbling	$O(\log \ell)$	$O(\log \ell)$

Each pebble stores one value x_i of the hash chain.

Initial positions of pebbles \bullet , for $\ell = 16$:

$\bullet \rightarrow \cdot \rightarrow \bullet \rightarrow \cdot \rightarrow \cdot \rightarrow \cdot \rightarrow \cdot \rightarrow \bullet \rightarrow \cdot \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \cdot$

Operations on pebbles:

- **clone** to create new pebble at same position
- **move** one position to the right (using one application of H)

See www.win.tue.nl/~berry/pebbling/ for optimal pebbling (incl. sample code).

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	--------

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•						•				•		•	•		x_{15}

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•							•				•	•	•	•		x_{15}
2	•								•			•	•	•			x_{14}

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•							•			•	•	•	•	•	•	x_{15}
2	•							•			•	•	•	•			x_{14}
3	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	x_{13}

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•							•				•	•	•	•	•	x_{15}
2	•							•				•	•				x_{14}
3	•	•	→	•				•	•	→	•	•	•	→	•		x_{13}
4	•	•	→	•	•			•	•	→	•	•	•				x_{12}

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•							•			•	•	•	•	•	•	x_{15}
2	•							•			•	•	•	•			x_{14}
3	•	•	→					•	•	→		•	•	→			x_{13}
4	•		→	•				•		→	•	•					x_{12}
5	•			→	•			•		•		•	→				x_{11}

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•							•				•	•	•	•	•	x_{15}
2	•							•				•	•				x_{14}
3	•	•	→					•	•	→		•	•	→			x_{13}
4	•		→	•				•		→	•	•	•				x_{12}
5	•			→	•			•		•	→	•					x_{11}
6	•				→	•		•			•						x_{10}

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		→	•	→				•		→	•	→				x_{12}
5	•			→	•				•		•	→	•				x_{11}
6	•				→	•			•		•						x_{10}
7	•					•	→	•	•	→	•						x_9

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		→	•	→				•		→	•	→				x_{12}
5	•				→	•			•		•	→	•				x_{11}
6	•					→	•		•		•						x_{10}
7	•						•	→	•		•						x_9
8	•							•	•	→	•						x_8

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		→	•	→				•		→	•	→				x_{12}
5	•			→	•				•		•	→	•				x_{11}
6	•				→	•			•			•					x_{10}
7	•					•	→	•		•		→	•				x_9
8	•						•	→	•								x_8
9	•							•		•	→	•					x_7

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		→	•	→				•		→	•	→				x_{12}
5	•			→	•				•			•	→				x_{11}
6	•				→	•			•			•					x_{10}
7	•					→	•	→		•		→					x_9
8	•						→	→	•								x_8
9	•							→	•								x_7
10	•								•	•							x_6

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		•	→					•		•	→		•			x_{12}
5	•			•	→				•		•	•	→				x_{11}
6	•				•	→			•		•						x_{10}
7	•					•	→		•		•	→					x_9
8	•						•	→	•								x_8
9	•							•	•	→							x_7
10	•								•								x_6
11	•									•	•	→					x_5

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		•	→					•		•	→		•			x_{12}
5	•			•	→				•		•	•	→				x_{11}
6	•				•	→			•		•						x_{10}
7	•					•	→		•		•	→					x_9
8	•						•	→	•								x_8
9	•							•		•	→						x_7
10	•								•								x_6
11	•	•	→						•		•	→					x_5
12	•			→	•	→											x_4

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		•	→					•		•	→		•			x_{12}
5	•			•	→				•		•	•	→				x_{11}
6	•				•	→			•		•						x_{10}
7	•					•	→		•		•	→					x_9
8	•						•	→	•								x_8
9	•							•	•	•	→						x_7
10	•								•								x_6
11	•	•	→						•		•	→					x_5
12	•		•	→						•							x_4
13	•			•	•	→											x_3

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		•	→					•		•	→		•			x_{12}
5	•			•	→				•		•	•	→				x_{11}
6	•				•	→			•		•						x_{10}
7	•					•	→		•		•	→					x_9
8	•						•	→	•								x_8
9	•							•	•	•	→						x_7
10	•								•		•						x_6
11	•	•	→						•		•	→					x_5
12	•		•	→					•								x_4
13	•			•	→												x_3
14	•				•												x_2

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		•	→					•		•	→		•			x_{12}
5	•			•	→				•		•	•	→				x_{11}
6	•				•	→			•		•						x_{10}
7	•					•	→		•		•	→					x_9
8	•						•	→	•								x_8
9	•							•	•	•	→						x_7
10	•								•		•						x_6
11	•	•	→						•		•	→					x_5
12	•		•	→					•								x_4
13	•			•	→												x_3
14	•				•												x_2
15	•					•	→										x_1

Pebbling Algorithm Visualization

Simple pebbling at **speed 1**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•	•	→						•	•	→			•	→		x_{13}
4	•		•	→					•		•	→		•			x_{12}
5	•			•	→				•		•	•	→				x_{11}
6	•				•	→			•		•						x_{10}
7	•					•	→		•		•	→					x_9
8	•						•	→	•								x_8
9	•							•	•	•	→						x_7
10	•								•		•						x_6
11	•	•	→						•		•	→					x_5
12	•		•	→					•								x_4
13	•			•	→												x_3
14	•				•												x_2
15	•		→														x_1
16	•																x_0

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	--------

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•						•				•		•	•	x_{15}	

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•							•			•		•	•	•	•	x_{15}
2	•								•			•		•	•	•	x_{14}

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•							•			•		•	•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•								•			•		•	•	•	x_{13}

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•							•			•	•	•	•	•	•	x_{15}
2	•							•			•	•	•				x_{14}
3	•							•			•	•	•				x_{13}
4	•							•	•		•						x_{12}

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•	•	•	•	•	x_{15}
2	•								•			•	•				x_{14}
3	•								•			•	•				x_{13}
4	•								•	•		•					x_{12}
5	•								•	•	•	•					x_{11}

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•	•	•	•	•	x_{15}
2	•								•			•	•				x_{14}
3	•								•			•		•			x_{13}
4	•								•		•						x_{12}
5	•								•	•		•					x_{11}
6	•								•	•	•						x_{10}

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•								•			•		•			x_{13}
4	•								•		•						x_{12}
5	•								•	•		•					x_{11}
6	•		•						•	•	•						x_{10}
7	•			•					•	•							x_9

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•	•	•	•	•	x_{15}
2	•								•			•	•				x_{14}
3	•								•			•		•			x_{13}
4	•								•		•						x_{12}
5	•								•	•		•					x_{11}
6	•		•						•	•	•						x_{10}
7	•		•	•					•	•	•						x_9
8	•			•	•				•								x_8

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•								•			•		•			x_{13}
4	•								•		•						x_{12}
5	•								•		•		•				x_{11}
6	•		•						•		•						x_{10}
7	•			•					•		•						x_9
8	•				•				•								x_8
9	•					•			•								x_7

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•								•			•		•			x_{13}
4	•								•		•						x_{12}
5	•								•		•		•				x_{11}
6	•		•						•		•						x_{10}
7	•			•					•		•						x_9
8	•				•				•								x_8
9	•					•			•								x_7
10	•					•			•								x_6

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•								•			•		•			x_{13}
4	•								•		•						x_{12}
5	•								•		•		•				x_{11}
6	•		→						•		•						x_{10}
7	•			→					•		•						x_9
8	•				→				•								x_8
9	•					→			•								x_7
10	•						→		•								x_6
11	•							→	•								x_5

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•	•	•	•	•	x_{15}
2	•								•			•	•				x_{14}
3	•								•			•					x_{13}
4	•								•		•						x_{12}
5	•								•	•		•					x_{11}
6	•								•	•	•						x_{10}
7	•								•	•							x_9
8	•								•		•						x_8
9	•								•	•							x_7
10	•								•	•	•						x_6
11	•								•	•							x_5
12	•								•		•						x_4

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•	•	•	•	•	x_{15}
2	•								•			•	•				x_{14}
3	•								•			•					x_{13}
4	•								•		•						x_{12}
5	•								•	•		•					x_{11}
6	•		•						•		•						x_{10}
7	•			•					•	•							x_9
8	•				•				•								x_8
9	•				•				•		•						x_7
10	•					•			•								x_6
11	•					•	•										x_5
12	•		•				•										x_4
13	•	•															x_3

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•		•	•	•	x_{15}
2	•								•			•		•			x_{14}
3	•								•			•					x_{13}
4	•								•		•						x_{12}
5	•								•		•						x_{11}
6	•								•		•						x_{10}
7	•								•		•						x_9
8	•								•								x_8
9	•								•		•						x_7
10	•								•		•						x_6
11	•								•		•						x_5
12	•								•								x_4
13	•								•								x_3
14	•								•								x_2

Pebbling Algorithm Visualization

Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

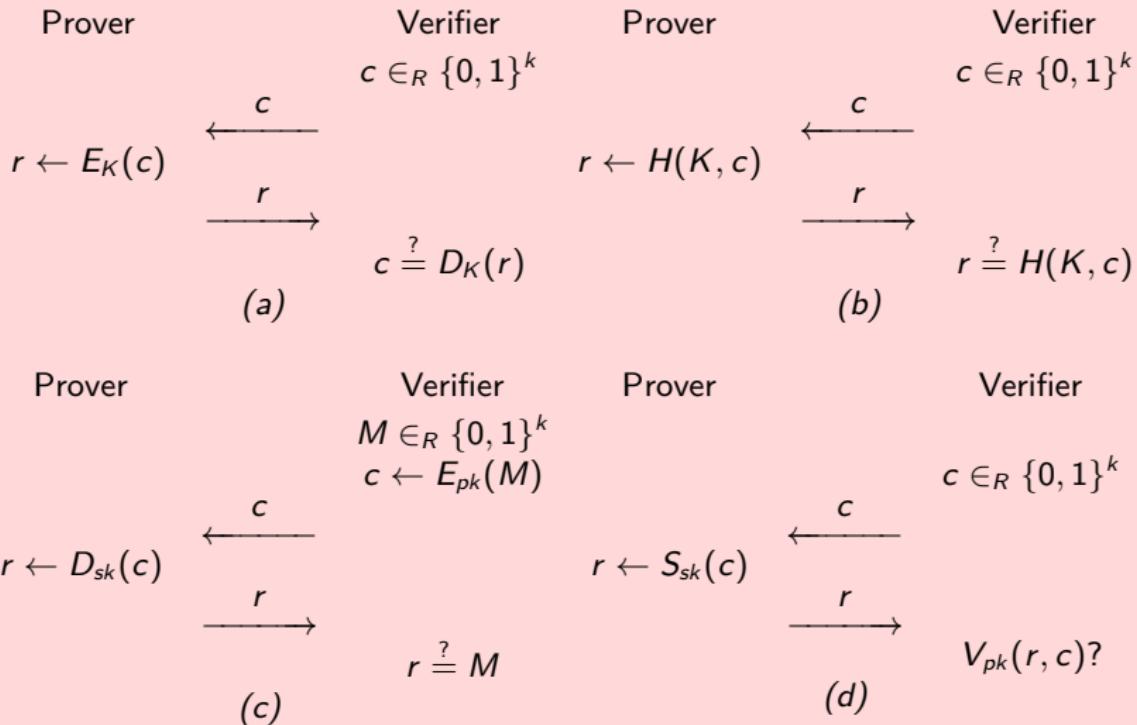
	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•	•	•	•	•	x_{15}
2	•								•			•	•				x_{14}
3	•								•			•					x_{13}
4	•								•		•						x_{12}
5	•								•	•	•						x_{11}
6	•								•	•	•						x_{10}
7	•								•	•	•						x_9
8	•								•		•						x_8
9	•								•	•	•						x_7
10	•								•	•	•						x_6
11	•								•	•	•						x_5
12	•								•	•	•						x_4
13	•								•	•	•						x_3
14	•								•	•	•						x_2
15	•								•	•	•						x_1

Pebbling Algorithm Visualization

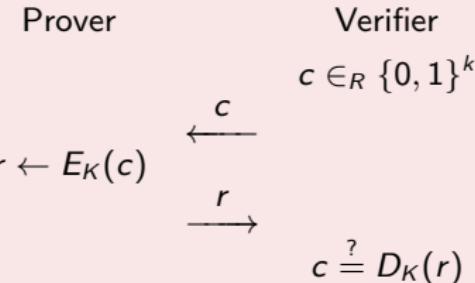
Space-efficient pebbling at **speed 2**, e.g., for $\ell = 16$:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	output
1	•								•			•	•	•	•	•	x_{15}
2	•								•			•	•				x_{14}
3	•								•			•					x_{13}
4	•								•		•						x_{12}
5	•								•	•	•						x_{11}
6	•								•	•	•						x_{10}
7	•								•	•	•						x_9
8	•								•		•						x_8
9	•								•	•	•						x_7
10	•								•	•	•						x_6
11	•								•	•	•						x_5
12	•								•	•	•						x_4
13	•								•	•	•						x_3
14	•								•		•						x_2
15	•								•		•						x_1
16	•								•		•						x_0

Figure 4.1 (Four basic challenge-response protocols)



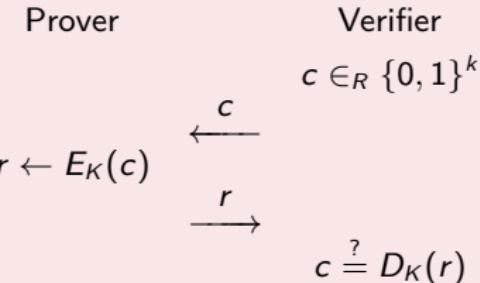
Basic challenge-response protocol (a)



Exercise 4.2

Consider the alternative protocol in which the verifier challenges the prover with $c = E_K(M)$, where $M \in_R \{0, 1\}^k$, and for which the prover is supposed to produce response $r = M$. Discuss eavesdropping attacks and cheating verifier attacks for this protocol.

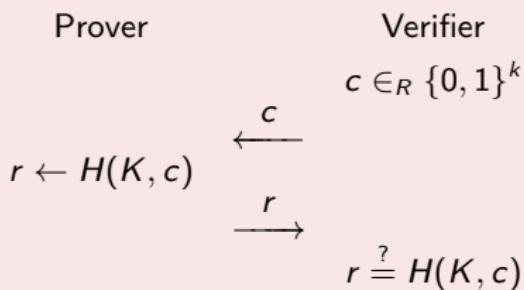
Basic challenge-response protocol (a)



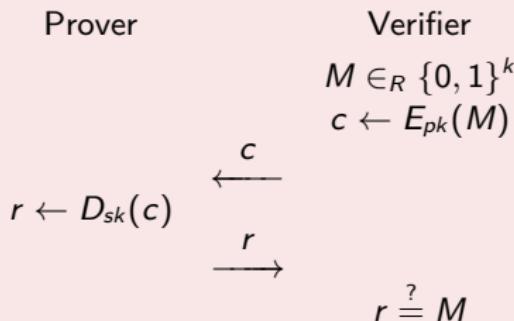
Exercise 4.2

Consider the alternative protocol in which the verifier challenges the prover with $c = E_K(M)$, where $M \in_R \{0, 1\}^k$, and for which the prover is supposed to produce response $r = M$. Discuss eavesdropping attacks and cheating verifier attacks for this protocol.

Basic challenge-response protocol (b)



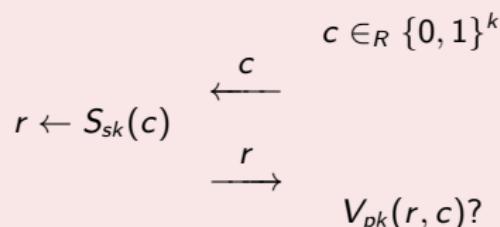
Basic challenge-response protocol (c)



Basic challenge-response protocol (d)

Prover

Verifier



4 Identification Protocols

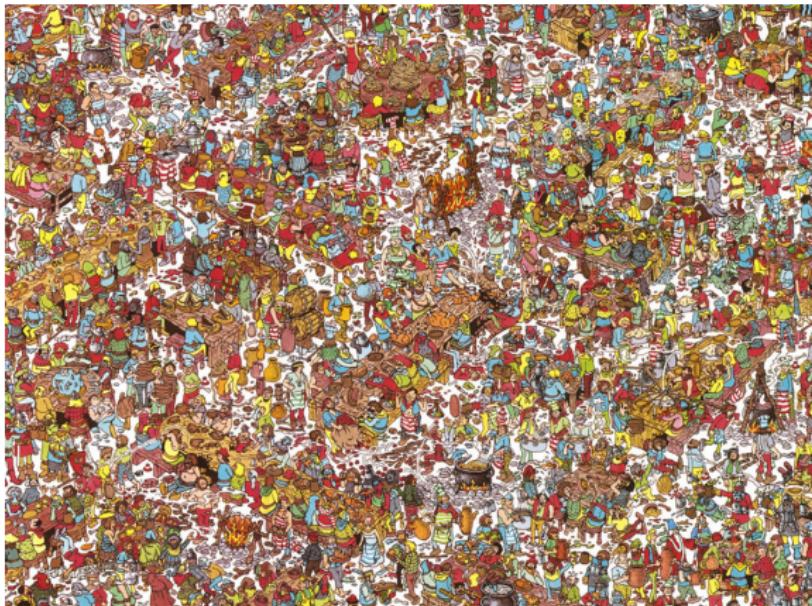
- 4.1 Definitions
 - 4.2 Password-based Schemes
 - 4.3 One-Way Hash Chains
 - 4.4 Basic Challenge-Response Protocols
 - 4.5 Zero-knowledge Identification Protocols
 - 4.6 Witness Hiding Identification Protocols
- 4.5.1 Schnorr Zero-knowledge Protocol
 - 4.5.2 Schnorr Protocol
 - 4.5.3 Guillou-Quisquater Protocol

Where's Waldo?



Find

in



Click this [PowerPoint slide show](#).

Zero-knowledge Identification Scheme

Registration. \mathcal{P} generates public key pair, keeps private key to itself, and gives public key to \mathcal{V} .

Identification. \mathcal{P} engages in zero-knowledge proof to convince \mathcal{V} of knowledge of private key.

Soundness property (security for \mathcal{V}): cheating \mathcal{P}^* not knowing private key does not succeed in convincing \mathcal{V} .

Zero-knowledge property (security for \mathcal{P}): cheating \mathcal{V}^* does not learn anything useful about private key from interacting with \mathcal{P} .

Discrete log setting $\langle g \rangle$.

Prover registered public key h , and knows private key x satisfying $y = g^x$.

Figure 4.2 (Schnorr's zero-knowledge protocol)

Prover

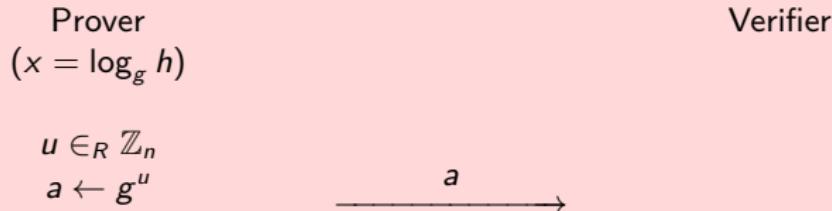
$$(x = \log_g h)$$

Verifier

Discrete log setting $\langle g \rangle$.

Prover registered public key h , and knows private key x satisfying $y = g^x$.

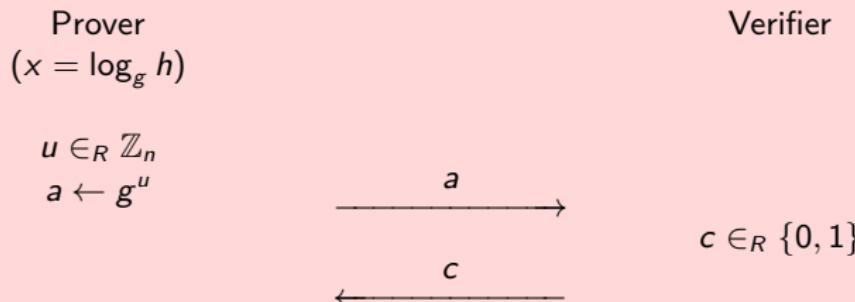
Figure 4.2 (Schnorr's zero-knowledge protocol)



Discrete log setting $\langle g \rangle$.

Prover registered public key h , and knows private key x satisfying $y = g^x$.

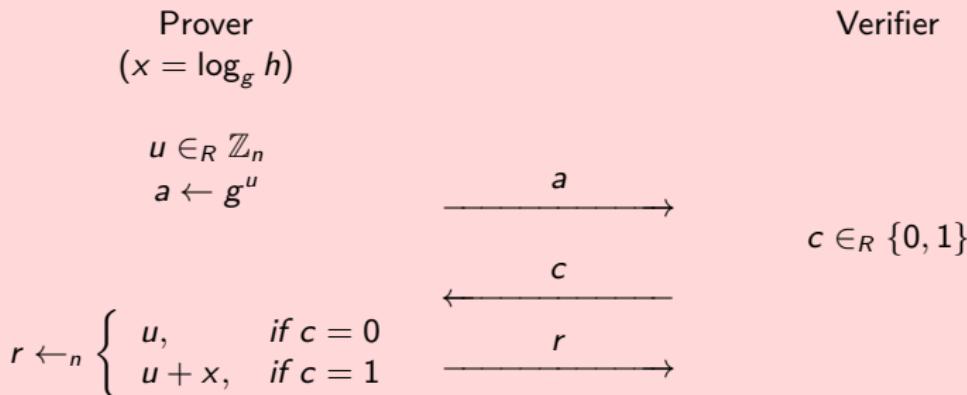
Figure 4.2 (Schnorr's zero-knowledge protocol)



Discrete log setting $\langle g \rangle$.

Prover registered public key h , and knows private key x satisfying $y = g^x$.

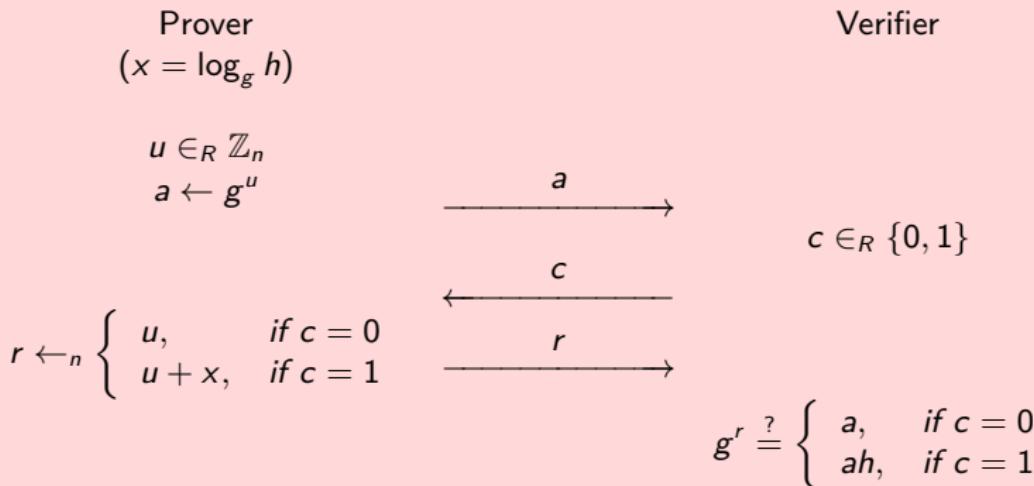
Figure 4.2 (Schnorr's zero-knowledge protocol)



Discrete log setting $\langle g \rangle$.

Prover registered public key h , and knows private key x satisfying $y = g^x$.

Figure 4.2 (Schnorr's zero-knowledge protocol)



Soundness

Soundness

Only prover \mathcal{P} knowing private key $x = \log_g h$ succeeds.

Cheating prover \mathcal{P}^* can do one of the following:

- Either, prepare for $c = 0$ by setting $a = g^u$ and using response $r = u$.
- Or, prepare for $c = 1$ by setting $a = g^u/h$ and using response $r = u$.

Verification $g^r = ah^c$ holds in both cases.

50% success probability for \mathcal{P}^*

But \mathcal{P}^* cannot do better than 50%!

Soundness

Suppose \mathcal{P}^* can answer both challenges $c = 0$ and $c = 1$ correctly, after sending announcement a to \mathcal{V} .

Hence \mathcal{P}^* can compute responses r_0 and r_1 such that:

$$g^{r_0} = a, \quad g^{r_1} = ah.$$

This implies

$$h = g^{r_1 - r_0}.$$

But then \mathcal{P}^* actually knows x , since $x = r_1 - r_0 \bmod n$ holds!

Zero-Knowledge

Cheating verifier \mathcal{V}^* may run protocol (polynomially) many times with \mathcal{P} , to obtain conversations $(a; c; r)$.

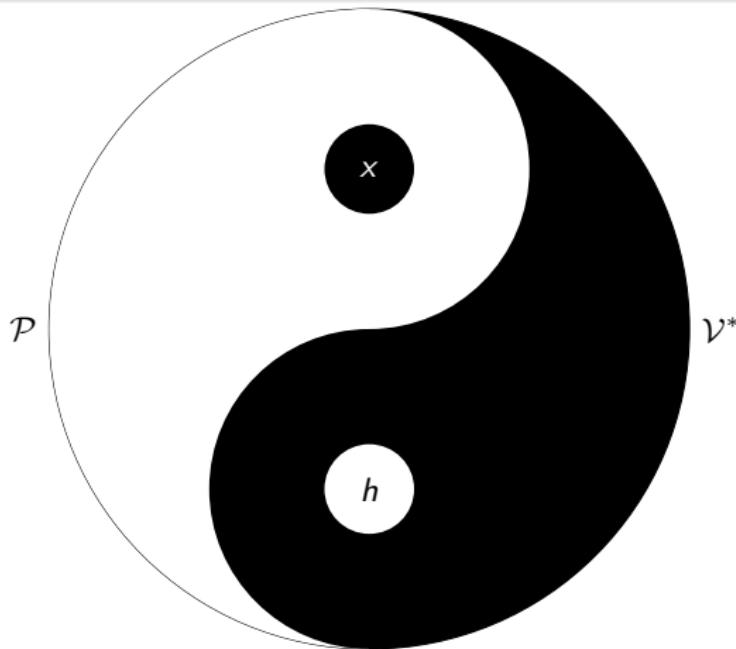
Simulation paradigm

\mathcal{V}^* can efficiently generate (**simulate**) these conversations **on its own**, given only public key h —and no access to prover \mathcal{P} who knows $x = \log_g h$.

Identification protocol is **zero-knowledge** if **simulator S** exists which, given public key h , generates conversations indistinguishable from conversations of real protocol runs between \mathcal{P} and \mathcal{V}^* .

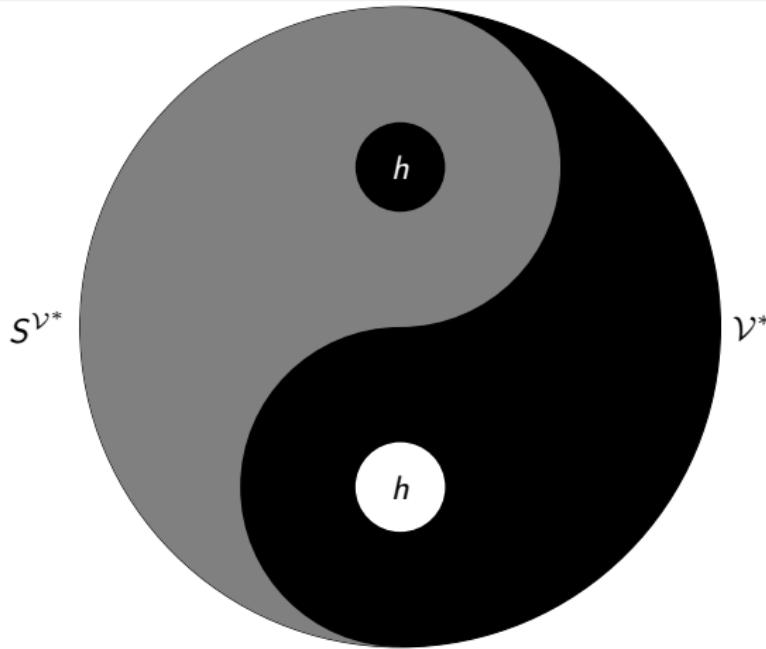
Usually, simulator S is using \mathcal{V}^* only as a rewritable black-box.
(I.e., usually, no need for S to know how \mathcal{V}^* operates internally.)

Simulation Paradigm



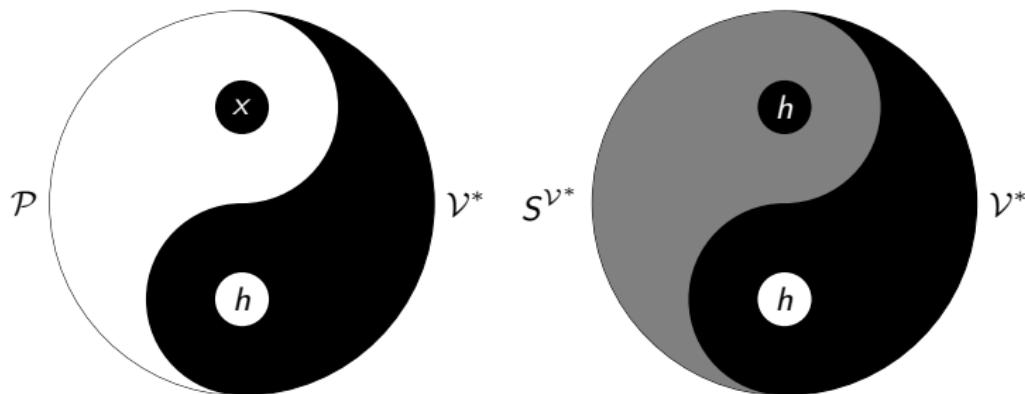
Real protocol runs between prover \mathcal{P} and cheating verifier \mathcal{V}^* ,
 \mathcal{P} using private key x

Simulation Paradigm



Simulated protocol runs between simulator S^{V^*} and cheating verifier V^* ,
 S^{V^*} using only public key h

Simulation Paradigm



\mathcal{V}^* cannot distinguish between protocol with \mathcal{P} and protocol with $S^{\mathcal{V}^*}$
⇒ protocol is zero-knowledge w.r.t. private key x

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ➊ $u \in_R \mathbb{Z}_n$
- ➋ $a \leftarrow g^u$
- ➌ $c \in_R \{0, 1\}$
- ➍ $r \leftarrow_n u + cx$
- ➎ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ➊ $c \in_R \{0, 1\}$
- ➋ $r \in_R \mathbb{Z}_n$
- ➌ $a \leftarrow g^r h^{-c}$
- ➍ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ **output** $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ **output** $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Simulation of Schnorr's Protocol

Real conversations

Input: **private key** x

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ $c \in_R \{0, 1\}$
- ④ $r \leftarrow_n u + cx$
- ⑤ output $(a; c; r)$

Simulated conversations

Input: **public key** h

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ output $(a; c; r)$

All real and simulated conversations $(a; c; r)$ are accepting, as $g^r = ah^c$ holds.

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **real conversations**:

$$\Pr[(a; c; r) = (A; C; R)]$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **real conversations**:

$$\Pr[(a; c; r) = (A; C; R)] = \Pr[a = A, c = C, r = R]$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **real conversations**:

$$\begin{aligned}\Pr[(a; c; r) = (A; C; R)] &= \Pr[a = A, c = C, r = R] \\ &= \Pr[u = \log_g A, c = C, u + cx = \log_g A + C \log_g h]\end{aligned}$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **real conversations**:

$$\begin{aligned}\Pr[(a; c; r) = (A; C; R)] &= \Pr[a = A, c = C, r = R] \\ &= \Pr[u = \log_g A, c = C, u + cx = \log_g A + C \log_g h] \\ &= \Pr[u = \log_g A, c = C]\end{aligned}$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **real conversations**:

$$\begin{aligned} \Pr[(a; c; r) = (A; C; R)] &= \Pr[a = A, c = C, r = R] \\ &= \Pr[u = \log_g A, c = C, u + cx = \log_g A + C \log_g h] \\ &= \Pr[u = \log_g A, c = C] \\ &= \Pr[u = \log_g A] \Pr[c = C] \end{aligned}$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **real conversations**:

$$\begin{aligned} \Pr[(a; c; r) = (A; C; R)] &= \Pr[a = A, c = C, r = R] \\ &= \Pr[u = \log_g A, c = C, u + cx = \log_g A + C \log_g h] \\ &= \Pr[u = \log_g A, c = C] \\ &= \Pr[u = \log_g A] \Pr[c = C] \\ &= \frac{1}{n} \frac{1}{2} = \frac{1}{2n} \end{aligned}$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **simulated conversations**:

$$\Pr[(a; c; r) = (A; C; R)]$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **simulated conversations**:

$$\Pr[(a; c; r) = (A; C; R)] = \Pr[a = A, c = C, r = R]$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **simulated conversations**:

$$\begin{aligned}\Pr[(a; c; r) = (A; C; R)] &= \Pr[a = A, c = C, r = R] \\ &= \Pr[g^r h^{-c} = g^R h^{-C}, c = C, r = R]\end{aligned}$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **simulated conversations**:

$$\begin{aligned}\Pr[(a; c; r) = (A; C; R)] &= \Pr[a = A, c = C, r = R] \\ &= \Pr[g^r h^{-c} = g^R h^{-C}, c = C, r = R] \\ &= \Pr[c = C, r = R]\end{aligned}$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **simulated conversations**:

$$\begin{aligned}\Pr[(a; c; r) = (A; C; R)] &= \Pr[a = A, c = C, r = R] \\ &= \Pr[g^r h^{-c} = g^R h^{-C}, c = C, r = R] \\ &= \Pr[c = C, r = R] \\ &= \Pr[c = C] \Pr[r = R]\end{aligned}$$

Why Are These Distributions Identical?

Consider fixed key pair $(h; x)$.

For fixed c , 1-to-1 correspondence between $u = \log_g a$ in real conversations and r in simulated conversations, as $r = u + cx$ holds in both cases.

Each accepting conversation $(a; c; r)$ occurs with probability $\frac{1}{2n}$ in both cases.

More precisely, let $(A; C; R)$ be any accepting conversation, hence $g^R = Ah^C$.

For **simulated conversations**:

$$\begin{aligned} \Pr[(a; c; r) = (A; C; R)] &= \Pr[a = A, c = C, r = R] \\ &= \Pr[g^r h^{-c} = g^R h^{-C}, c = C, r = R] \\ &= \Pr[c = C, r = R] \\ &= \Pr[c = C] \Pr[r = R] \\ &= \frac{1}{2} \frac{1}{n} = \frac{1}{2n} \end{aligned}$$

Passive Impersonation Attacks

Security against passive impersonation attacks extends to any eavesdropper.

Remark 4.3

Honest-verifier zero-knowledge reduces any **passive** impersonation attack to a key-only attack: eavesdropping conversations does not yield anything about prover's private key beyond what can be deduced from corresponding public key.

Eavesdropped conversations $(a; c; r)$ for honest runs of Schnorr's protocol can be simulated as well given public key h only.

But, eavesdropper may learn something new!

From a single conversation $(a; c; r)$ with $c \neq 0$,
eavesdropper may **recover prover's public key** as $h = (g^r/a)^{1/c}$
(thus, potentially **establishing prover's identity**).

Passive Impersonation Attacks

Security against passive impersonation attacks extends to any eavesdropper.

Remark 4.3

Honest-verifier zero-knowledge reduces any **passive** impersonation attack to a key-only attack: eavesdropping conversations does not yield anything about prover's private key beyond what can be deduced from corresponding public key.

Eavesdropped conversations $(a; c; r)$ for honest runs of Schnorr's protocol can be simulated as well given public key h only.

But, eavesdropper may learn something new!

From a single conversation $(a; c; r)$ with $c \neq 0$,
eavesdropper may **recover prover's public key** as $h = (g^r/a)^{1/c}$
(thus, potentially **establishing prover's identity**).

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

$$\textcircled{1} \quad u \in_R \mathbb{Z}_n$$

$$\textcircled{2} \quad a \leftarrow g^u$$

\textcircled{3} send a to \mathcal{V}^*

\textcircled{4} receive $c \in \{0, 1\}$ from \mathcal{V}^*

$$\textcircled{5} \quad r \leftarrow_n u + cx$$

\textcircled{6} output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

$$\textcircled{1} \quad c \in_R \{0, 1\}$$

$$\textcircled{2} \quad r \in_R \mathbb{Z}_n$$

$$\textcircled{3} \quad a \leftarrow g^r h^{-c}$$

\textcircled{4} send a to \mathcal{V}^*

\textcircled{5} receive $c' \in \{0, 1\}$ from \mathcal{V}^*

\textcircled{6} if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1

\textcircled{7} output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

$$\textcircled{1} \quad u \in_R \mathbb{Z}_n$$

$$\textcircled{2} \quad a \leftarrow g^u$$

\textcircled{3} send a to \mathcal{V}^*

\textcircled{4} receive $c \in \{0, 1\}$ from \mathcal{V}^*

$$\textcircled{5} \quad r \leftarrow_n u + cx$$

\textcircled{6} output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

$$\textcircled{1} \quad c \in_R \{0, 1\}$$

$$\textcircled{2} \quad r \in_R \mathbb{Z}_n$$

$$\textcircled{3} \quad a \leftarrow g^r h^{-c}$$

\textcircled{4} send a to \mathcal{V}^*

\textcircled{5} receive $c' \in \{0, 1\}$ from \mathcal{V}^*

\textcircled{6} if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1

\textcircled{7} output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

$$\textcircled{1} \quad u \in_R \mathbb{Z}_n$$

$$\textcircled{2} \quad a \leftarrow g^u$$

③ send a to \mathcal{V}^*

④ receive $c \in \{0, 1\}$ from \mathcal{V}^*

$$\textcircled{5} \quad r \leftarrow_n u + cx$$

⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

$$\textcircled{1} \quad c \in_R \{0, 1\}$$

$$\textcircled{2} \quad r \in_R \mathbb{Z}_n$$

$$\textcircled{3} \quad a \leftarrow g^r h^{-c}$$

④ send a to \mathcal{V}^*

⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*

⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1

⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ **output $(a; c; r)$**

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ **output $(a; c; r)$**

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

$$\textcircled{1} \quad u \in_R \mathbb{Z}_n$$

$$\textcircled{2} \quad a \leftarrow g^u$$

\textcircled{3} send a to \mathcal{V}^*

\textcircled{4} receive $c \in \{0, 1\}$ from \mathcal{V}^*

$$\textcircled{5} \quad r \leftarrow_n u + cx$$

\textcircled{6} output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

$$\textcircled{1} \quad c \in_R \{0, 1\}$$

$$\textcircled{2} \quad r \in_R \mathbb{Z}_n$$

$$\textcircled{3} \quad a \leftarrow g^r h^{-c}$$

\textcircled{4} send a to \mathcal{V}^*

\textcircled{5} receive $c' \in \{0, 1\}$ from \mathcal{V}^*

\textcircled{6} if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1

\textcircled{7} output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ **send a to \mathcal{V}^***
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.



Simulation of Schnorr's Protocol

Recall \mathcal{V}^* is p.p.t. algorithm (Turing machine).

Rewinding \mathcal{V}^* : going back to a previous configuration

(configuration = state + tape contents + position of read/write heads).

Real conversations

Input: **private key x**

Output: conversation $(a; c; r)$

- ① $u \in_R \mathbb{Z}_n$
- ② $a \leftarrow g^u$
- ③ send a to \mathcal{V}^*
- ④ receive $c \in \{0, 1\}$ from \mathcal{V}^*
- ⑤ $r \leftarrow_n u + cx$
- ⑥ output $(a; c; r)$

Simulated conversations

Input: **public key h**

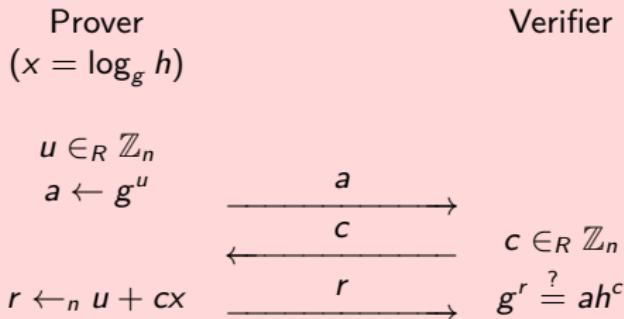
Output: conversation $(a; c; r)$

- ① $c \in_R \{0, 1\}$
- ② $r \in_R \mathbb{Z}_n$
- ③ $a \leftarrow g^r h^{-c}$
- ④ send a to \mathcal{V}^*
- ⑤ receive $c' \in \{0, 1\}$ from \mathcal{V}^*
- ⑥ if $c \neq c'$ rewind \mathcal{V}^* to point prior to receiving a and go to step 1
- ⑦ output $(a; c; r)$

Sequential iterations can be simulated as well.

Cheating probability $1/2^k$ for k iterations.

Figure 4.3 (Schnorr's identification protocol)



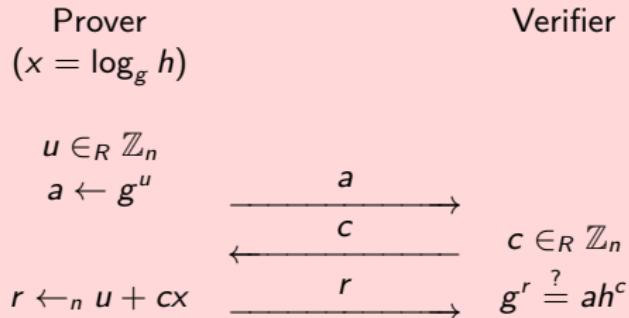
Soundness: cheating probability down to $1/n$.

Zero-knowledge: only for **honest** verifier \mathcal{V} .

Identical distributions of real/simulated conversations:

$$\begin{aligned} & \{(a; c; r) : u, c \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ & \{(a; c; r) : c, r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Figure 4.3 (Schnorr's identification protocol)



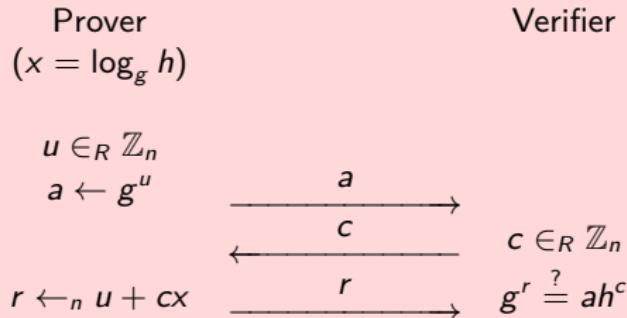
Soundness: cheating probability down to $1/n$.

Zero-knowledge: only for **honest** verifier \mathcal{V} .

Identical distributions of real/simulated conversations:

$$\begin{aligned} & \{(a; c; r) : u, c \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ & \{(a; c; r) : c, r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Figure 4.3 (Schnorr's identification protocol)



Soundness: cheating probability down to $1/n$.

Zero-knowledge: only for **honest** verifier \mathcal{V} .

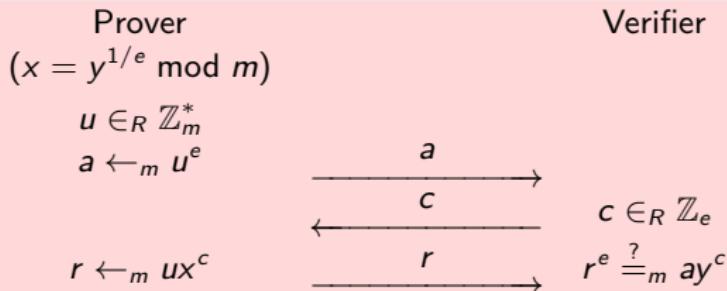
Identical distributions of real/simulated conversations:

$$\begin{aligned} & \{(a; c; r) : u, c \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ & \{(a; c; r) : c, r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

RSA setting: RSA modulus m , public exponent large prime e .

Prover registered public key y , and knows private key x s.t. $y = x^e \bmod m$.

Figure 4.4 (Guillou-Quisquater's identification protocol)



Soundness: cheating probability $1/e$.

Zero-knowledge: only for **honest** verifier \mathcal{V} .

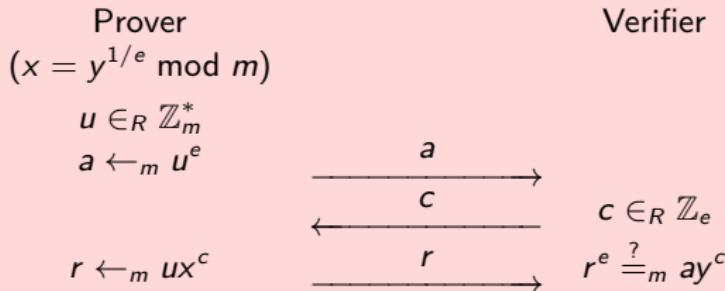
Identical distributions of real/simulated conversations:

$$\begin{aligned} & \{(a; c; r) : u \in_R \mathbb{Z}_m^*; c \in_R \mathbb{Z}_e; a \leftarrow_m u^e; r \leftarrow_m ux^c\}, \\ & \{(a; c; r) : c \in_R \mathbb{Z}_e; r \in_R \mathbb{Z}_m^*; a \leftarrow_m r^e y^{-c}\}. \end{aligned}$$

RSA setting: RSA modulus m , public exponent large prime e .

Prover registered public key y , and knows private key x s.t. $y = x^e \bmod m$.

Figure 4.4 (Guillou-Quisquater's identification protocol)



Soundness: cheating probability $1/e$.

Zero-knowledge: only for **honest** verifier \mathcal{V} .

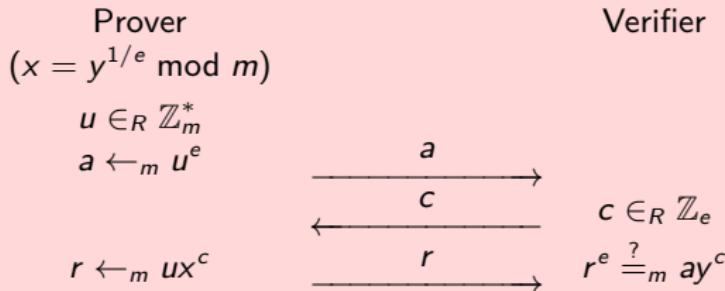
Identical distributions of real/simulated conversations:

$$\begin{aligned} & \{(a; c; r) : u \in_R \mathbb{Z}_m^*; c \in_R \mathbb{Z}_e; a \leftarrow_m u^e; r \leftarrow_m ux^c\}, \\ & \{(a; c; r) : c \in_R \mathbb{Z}_e; r \in_R \mathbb{Z}_m^*; a \leftarrow_m r^e y^{-c}\}. \end{aligned}$$

RSA setting: RSA modulus m , public exponent large prime e .

Prover registered public key y , and knows private key x s.t. $y = x^e \bmod m$.

Figure 4.4 (Guillou-Quisquater's identification protocol)



Soundness: cheating probability $1/e$.

Zero-knowledge: only for **honest** verifier \mathcal{V} .

Identical distributions of real/simulated conversations:

$$\begin{aligned}
 & \{(a; c; r) : u \in_R \mathbb{Z}_m^*; c \in_R \mathbb{Z}_e; a \leftarrow_m u^e; r \leftarrow_m ux^c\}, \\
 & \{(a; c; r) : c \in_R \mathbb{Z}_e; r \in_R \mathbb{Z}_m^*; a \leftarrow_m r^e y^{-c}\}.
 \end{aligned}$$

4 Identification Protocols
4.1 Definitions
4.2 Password-based Schemes
4.3 One-Way Hash Chains
4.4 Basic Challenge-Response Protocols
4.5 Zero-knowledge Identification Protocols
4.6 Witness Hiding Identification Protocols

4.5.1 Schnorr Zero-knowledge Protocol
4.5.2 Schnorr Protocol
4.5.3 Guillou-Quisquater Protocol

Soundness of Guillou-Quisquater's Protocol

Assume prover able to produce accepting $(a; c; r), (a; c'; r')$, with $c \neq c'$.
 Then (modulo m):

$$\begin{aligned} r^e &= ay^c, & r'^e &= ay^{c'} \\ \Rightarrow \quad (r/r')^e &= y^{c-c'} \end{aligned}$$

1st attempt: rewrite as $y = (r/r')^{e/(c-c')}$.

But inversion of $c - c'$ modulo $\phi(m)$ **cannot be computed efficiently**, as $\phi(m)$ is not known.

2nd attempt: $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$.
 Extended Euclidean algorithm yields $s, t \in \mathbb{Z}$ satisfying $se + t(c - c') = 1$.
 Raise both sides to power t :

$$\begin{aligned} (r/r')^{te} &= y^{t(c-c')} = y^{1-se} \\ \Rightarrow \quad y &= (y^s(r/r')^t)^e \end{aligned}$$

Private key $x \leftarrow_m y^s(r/r')^t$ thus known to prover.

Soundness of Guillou-Quisquater's Protocol

Assume prover able to produce accepting $(a; c; r), (a; c'; r')$, with $c \neq c'$.
 Then (modulo m):

$$\begin{aligned} r^e &= ay^c, & r'^e &= ay^{c'} \\ \Rightarrow \quad (r/r')^e &= y^{c-c'} \end{aligned}$$

1st attempt: rewrite as $y = (r/r')^{e/(c-c')}$.

But inversion of $c - c'$ modulo $\phi(m)$ **cannot be computed efficiently**, as $\phi(m)$ is not known.

2nd attempt: $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$.
 Extended Euclidean algorithm yields $s, t \in \mathbb{Z}$ satisfying $se + t(c - c') = 1$.
 Raise both sides to power t :

$$\begin{aligned} (r/r')^{te} &= y^{t(c-c')} = y^{1-se} \\ \Rightarrow \quad y &= (y^s(r/r')^t)^e \end{aligned}$$

Private key $x \leftarrow_m y^s(r/r')^t$ thus known to prover.

Soundness of Guillou-Quisquater's Protocol

Assume prover able to produce accepting $(a; c; r), (a; c'; r')$, with $c \neq c'$.
 Then (modulo m):

$$\begin{aligned} r^e &= ay^c, & r'^e &= ay^{c'} \\ \Rightarrow \quad (r/r')^e &= y^{c-c'} \end{aligned}$$

1st attempt: rewrite as $y = (r/r')^{e/(c-c')}$.

But inversion of $c - c'$ modulo $\phi(m)$ **cannot be computed efficiently**, as $\phi(m)$ is not known.

2nd attempt: $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$.
 Extended Euclidean algorithm yields $s, t \in \mathbb{Z}$ satisfying $se + t(c - c') = 1$.
 Raise both sides to power t :

$$\begin{aligned} (r/r')^{te} &= y^{t(c-c')} = y^{1-se} \\ \Rightarrow \quad y &= (y^s(r/r')^t)^e \end{aligned}$$

Private key $x \leftarrow_m y^s(r/r')^t$ thus known to prover.

Soundness of Guillou-Quisquater's Protocol

Assume prover able to produce accepting $(a; c; r), (a; c'; r')$, with $c \neq c'$.
 Then (modulo m):

$$\begin{aligned} r^e &= ay^c, & r'^e &= ay^{c'} \\ \Rightarrow \quad (r/r')^e &= y^{c-c'} \end{aligned}$$

1st attempt: rewrite as $y = (r/r')^{e/(c-c')}$.

But inversion of $c - c'$ modulo $\phi(m)$ **cannot be computed efficiently**, as $\phi(m)$ is not known.

2nd attempt: $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$.
 Extended Euclidean algorithm yields $s, t \in \mathbb{Z}$ satisfying $se + t(c - c') = 1$.
 Raise both sides to power t :

$$\begin{aligned} (r/r')^{te} &= y^{t(c-c')} = y^{1-se} \\ \Rightarrow \quad y &= (y^s(r/r')^t)^e \end{aligned}$$

Private key $x \leftarrow_m y^s(r/r')^t$ thus known to prover.

Soundness of Guillou-Quisquater's Protocol

Assume prover able to produce accepting $(a; c; r), (a; c'; r')$, with $c \neq c'$.
 Then (modulo m):

$$\begin{aligned} r^e &= ay^c, & r'^e &= ay^{c'} \\ \Rightarrow \quad (r/r')^e &= y^{c-c'} \end{aligned}$$

1st attempt: rewrite as $y = (r/r')^{e/(c-c')}$.

But inversion of $c - c'$ modulo $\phi(m)$ **cannot be computed efficiently**, as $\phi(m)$ is not known.

2nd attempt: $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$.

Extended Euclidean algorithm yields $s, t \in \mathbb{Z}$ satisfying $se + t(c - c') = 1$.
 Raise both sides to power t :

$$\begin{aligned} (r/r')^{te} &= y^{t(c-c')} = y^{1-se} \\ \Rightarrow \quad y &= (y^s(r/r')^t)^e \end{aligned}$$

Private key $x \leftarrow_m y^s(r/r')^t$ thus known to prover.

Soundness of Guillou-Quisquater's Protocol

Assume prover able to produce accepting $(a; c; r), (a; c'; r')$, with $c \neq c'$.
 Then (modulo m):

$$\begin{aligned} r^e &= ay^c, & r'^e &= ay^{c'} \\ \Rightarrow \quad (r/r')^e &= y^{c-c'} \end{aligned}$$

1st attempt: rewrite as $y = (r/r')^{e/(c-c')}$.

But inversion of $c - c'$ modulo $\phi(m)$ **cannot be computed efficiently**, as $\phi(m)$ is not known.

2nd attempt: $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$.

Extended Euclidean algorithm yields $s, t \in \mathbb{Z}$ satisfying $se + t(c - c') = 1$.

Raise both sides to power t :

$$\begin{aligned} (r/r')^{te} &= y^{t(c-c')} = y^{1-se} \\ \Rightarrow \quad y &= (y^s(r/r')^t)^e \end{aligned}$$

Private key $x \leftarrow_m y^s(r/r')^t$ thus known to prover.

Soundness of Guillou-Quisquater's Protocol

Assume prover able to produce accepting $(a; c; r), (a; c'; r')$, with $c \neq c'$.
 Then (modulo m):

$$\begin{aligned} r^e &= ay^c, & r'^e &= ay^{c'} \\ \Rightarrow \quad (r/r')^e &= y^{c-c'} \end{aligned}$$

1st attempt: rewrite as $y = (r/r')^{e/(c-c')}$.

But inversion of $c - c'$ modulo $\phi(m)$ **cannot be computed efficiently**, as $\phi(m)$ is not known.

2nd attempt: $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$.
 Extended Euclidean algorithm yields $s, t \in \mathbb{Z}$ satisfying $se + t(c - c') = 1$.
 Raise both sides to power t :

$$\begin{aligned} (r/r')^{te} &= y^{t(c-c')} = y^{1-se} \\ \Rightarrow \quad y &= (y^s(r/r')^t)^e \end{aligned}$$

Private key $x \leftarrow_m y^s(r/r')^t$ thus known to prover.

Soundness of Guillou-Quisquater's Protocol

Assume prover able to produce accepting $(a; c; r), (a; c'; r')$, with $c \neq c'$.
 Then (modulo m):

$$\begin{aligned} r^e &= ay^c, & r'^e &= ay^{c'} \\ \Rightarrow \quad (r/r')^e &= y^{c-c'} \end{aligned}$$

1st attempt: rewrite as $y = (r/r')^{e/(c-c')}$.

But inversion of $c - c'$ modulo $\phi(m)$ **cannot be computed efficiently**, as $\phi(m)$ is not known.

2nd attempt: $\gcd(e, c - c') = 1$, since e is prime and $c, c' \in \mathbb{Z}_e, c \neq c'$.
 Extended Euclidean algorithm yields $s, t \in \mathbb{Z}$ satisfying $se + t(c - c') = 1$.
 Raise both sides to power t :

$$\begin{aligned} (r/r')^{te} &= y^{t(c-c')} = y^{1-se} \\ \Rightarrow \quad y &= (y^s(r/r')^t)^e \end{aligned}$$

Private key $x \leftarrow_m y^s(r/r')^t$ thus known to prover.

Identity-Based Identification Scheme

Remark 4.4

Prover does not need to know factorization of m .

Therefore, users may share same modulus m .

Identity-based identification scheme: public keys determined by users' identities.

Set user \mathcal{A} 's public key as $y_{\mathcal{A}} = H(ID_{\mathcal{A}})$, where $ID_{\mathcal{A}}$ may consist of user's name and/or email address.

Trusted third party \mathcal{T} required to compute private key of each user.

Only \mathcal{T} needs to know factorization of m : to compute user \mathcal{A} 's private key as $x_{\mathcal{A}} = H(ID_{\mathcal{A}})^{1/e} \bmod m$.

Advantage: public keys need not be certified by a digital signature from \mathcal{T} .

Disadvantage: \mathcal{T} knows private key of every user; problem may be alleviated by distributing role of \mathcal{T} between many parties, using threshold cryptography.

Witness Hiding vs Zero-Knowledge

Schnorr's identification protocol:

- 1 iteration with $c \in_R \mathbb{Z}_n$: efficient, but only honest-verifier zero-knowledge
- k iterations with $c \in_R \{0, 1\}$: zero-knowledge, but not efficient

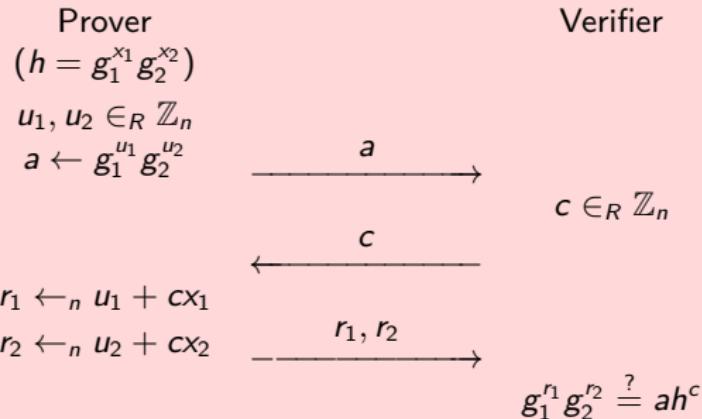
Witness hiding protocols: strike a balance between security and efficiency.

Identification protocol is **witness hiding** if cheating verifier is not able to obtain the **complete** prover's private key.

Witness hiding protocol is not necessarily zero-knowledge:
cheating verifier may be able to extract some **partial** information on private key,
but not sufficient for successful impersonation of the prover.

Private key (x_1, x_2) satisfies $h = g_1^{x_1} g_2^{x_2}$ for public key h .

Figure 4.5 (Okamoto's identification protocol)



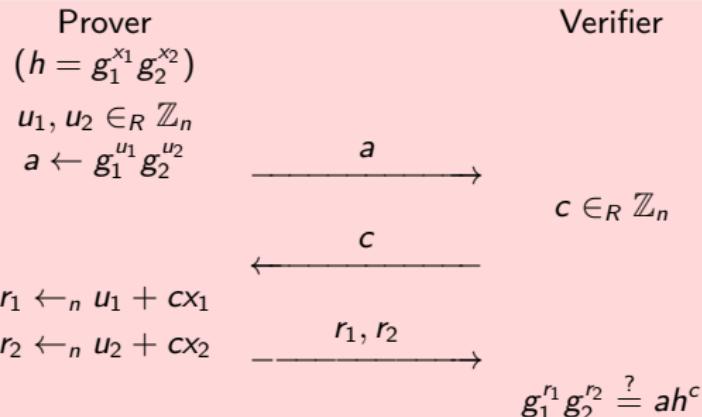
Assume $\log_{g_1} g_2$ not known to anybody (cf. Pedersen commitments).
 Okamoto's protocol is **witness hiding**:

cheating verifier \mathcal{V}^* cannot find “witness” (x_1, x_2) .

Note: not excluded that \mathcal{V}^* can find **partial** information on (x_1, x_2) !

Private key (x_1, x_2) satisfies $h = g_1^{x_1} g_2^{x_2}$ for public key h .

Figure 4.5 (Okamoto's identification protocol)



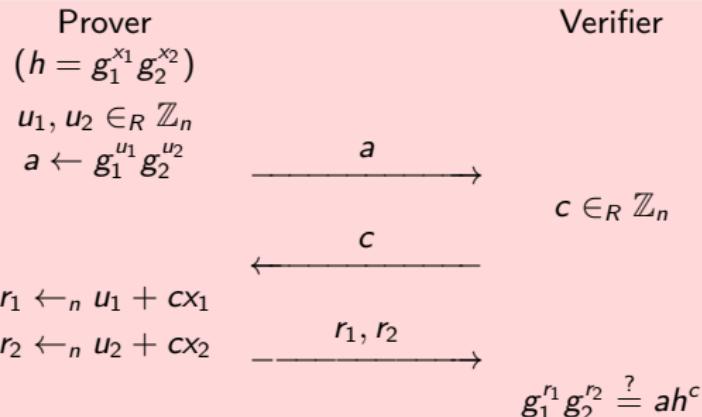
Assume $\log_{g_1} g_2$ not known to anybody (cf. Pedersen commitments).
 Okamoto's protocol is **witness hiding**:

cheating verifier \mathcal{V}^* cannot find “witness” (x_1, x_2) .

Note: not excluded that \mathcal{V}^* can find **partial** information on (x_1, x_2) !

Private key (x_1, x_2) satisfies $h = g_1^{x_1} g_2^{x_2}$ for public key h .

Figure 4.5 (Okamoto's identification protocol)

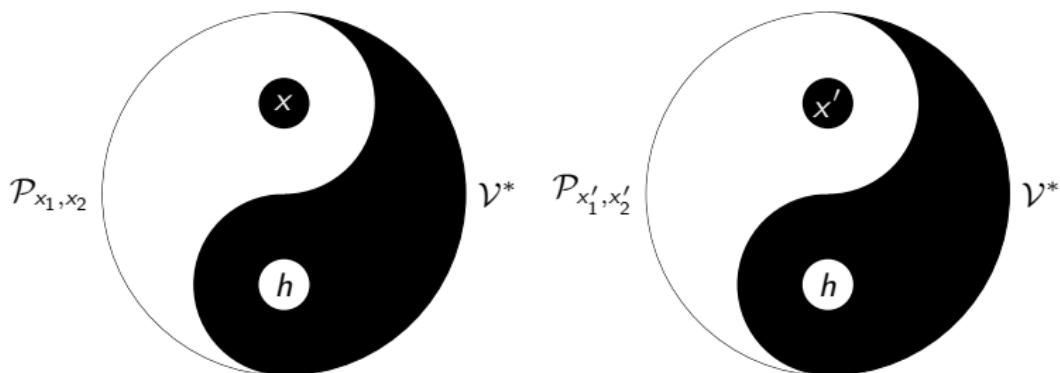


Assume $\log_{g_1} g_2$ not known to anybody (cf. Pedersen commitments).
 Okamoto's protocol is **witness hiding**:

cheating verifier \mathcal{V}^* cannot find “witness” (x_1, x_2) .

Note: not excluded that \mathcal{V}^* can find **partial** information on (x_1, x_2) !

Witness Indistinguishability



\mathcal{V}^* cannot distinguish between protocol with \mathcal{P}_{x_1, x_2} and protocol with $\mathcal{P}_{x'_1, x'_2}$
⇒ protocol is witness indistinguishable

Witness Indistinguishability of Okamoto's Protocol

Consider conversation $(a; c; r_1, r_2)$ between \mathcal{P}_{x_1, x_2} and \mathcal{V}^* , where $a = g_1^{u_1} g_2^{u_2}$.

Let (x'_1, x'_2) be any witness satisfying $h = g_1^{x'_1} g_2^{x'_2}$.

Then unique u'_1, u'_2 exist that yield same conversation $(a; c; r_1, r_2)$ for $\mathcal{P}_{x'_1, x'_2}$:

$$u'_1, u'_2 \leftarrow_n u_1 + c(x_1 - x'_1), u_2 + c(x_2 - x'_2)$$

Indeed:

$$\begin{aligned} a' &= g_1^{u'_1} g_2^{u'_2} = g_1^{u_1} g_2^{u_2} (g_1^{x_1} g_2^{x_2})^c / (g_1^{x'_1} g_2^{x'_2})^c = a \\ r'_1 &= u'_1 + cx'_1 = u_1 + c(x_1 - x'_1) + cx'_1 = r_1 \\ r'_2 &= u'_2 + cx'_2 = u_2 + c(x_2 - x'_2) + cx'_2 = r_2 \end{aligned}$$

Perfect witness indistinguishability as conversations with \mathcal{P}_{x_1, x_2} and conversations with $\mathcal{P}_{x'_1, x'_2}$ are distributed identically:

$$\{(a; c; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u_1} g_2^{u_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u_1 + cx_1, u_2 + cx_2\}$$

$$\{(a; c; r_1, r_2) : u'_1, u'_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u'_1} g_2^{u'_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u'_1 + cx'_1, u'_2 + cx'_2\}$$

Witness Indistinguishability of Okamoto's Protocol

Consider conversation $(a; c; r_1, r_2)$ between \mathcal{P}_{x_1, x_2} and \mathcal{V}^* , where $a = g_1^{u_1} g_2^{u_2}$.

Let (x'_1, x'_2) be any witness satisfying $h = g_1^{x'_1} g_2^{x'_2}$.

Then unique u'_1, u'_2 exist that yield same conversation $(a; c; r_1, r_2)$ for $\mathcal{P}_{x'_1, x'_2}$:

$$u'_1, u'_2 \leftarrow_n u_1 + c(x_1 - x'_1), u_2 + c(x_2 - x'_2)$$

Indeed:

$$\begin{aligned} a' &= g_1^{u'_1} g_2^{u'_2} = g_1^{u_1} g_2^{u_2} (g_1^{x_1} g_2^{x_2})^c / (g_1^{x'_1} g_2^{x'_2})^c = a \\ r'_1 &= u'_1 + cx'_1 = u_1 + c(x_1 - x'_1) + cx'_1 = r_1 \\ r'_2 &= u'_2 + cx'_2 = u_2 + c(x_2 - x'_2) + cx'_2 = r_2 \end{aligned}$$

Perfect witness indistinguishability as conversations with \mathcal{P}_{x_1, x_2} and conversations with $\mathcal{P}_{x'_1, x'_2}$ are distributed identically:

$$\{(a; c; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u_1} g_2^{u_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u_1 + cx_1, u_2 + cx_2\}$$

$$\{(a; c; r_1, r_2) : u'_1, u'_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u'_1} g_2^{u'_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u'_1 + cx'_1, u'_2 + cx'_2\}$$

Witness Indistinguishability of Okamoto's Protocol

Consider conversation $(a; c; r_1, r_2)$ between \mathcal{P}_{x_1, x_2} and \mathcal{V}^* , where $a = g_1^{u_1} g_2^{u_2}$.

Let (x'_1, x'_2) be any witness satisfying $h = g_1^{x'_1} g_2^{x'_2}$.

Then unique u'_1, u'_2 **exist** that yield same conversation $(a; c; r_1, r_2)$ for $\mathcal{P}_{x'_1, x'_2}$:

$$u'_1, u'_2 \leftarrow_n u_1 + c(x_1 - x'_1), u_2 + c(x_2 - x'_2)$$

Indeed:

$$a' = g_1^{u'_1} g_2^{u'_2} = g_1^{u_1} g_2^{u_2} (g_1^{x_1} g_2^{x_2})^c / (g_1^{x'_1} g_2^{x'_2})^c = a$$

$$r'_1 = u'_1 + cx'_1 = u_1 + c(x_1 - x'_1) + cx'_1 = r_1$$

$$r'_2 = u'_2 + cx'_2 = u_2 + c(x_2 - x'_2) + cx'_2 = r_2$$

Perfect witness indistinguishability as conversations with \mathcal{P}_{x_1, x_2} and conversations with $\mathcal{P}_{x'_1, x'_2}$ are distributed identically:

$$\{(a; c; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u_1} g_2^{u_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u_1 + cx_1, u_2 + cx_2\}$$

$$\{(a; c; r_1, r_2) : u'_1, u'_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u'_1} g_2^{u'_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u'_1 + cx'_1, u'_2 + cx'_2\}$$

Witness Indistinguishability of Okamoto's Protocol

Consider conversation $(a; c; r_1, r_2)$ between \mathcal{P}_{x_1, x_2} and \mathcal{V}^* , where $a = g_1^{u_1} g_2^{u_2}$.

Let (x'_1, x'_2) be any witness satisfying $h = g_1^{x'_1} g_2^{x'_2}$.

Then unique u'_1, u'_2 **exist** that yield same conversation $(a; c; r_1, r_2)$ for $\mathcal{P}_{x'_1, x'_2}$:

$$u'_1, u'_2 \leftarrow_n u_1 + c(x_1 - x'_1), u_2 + c(x_2 - x'_2)$$

Indeed:

$$\begin{aligned} a' &= g_1^{u'_1} g_2^{u'_2} = g_1^{u_1} g_2^{u_2} (g_1^{x_1} g_2^{x_2})^c / (g_1^{x'_1} g_2^{x'_2})^c = a \\ r'_1 &= u'_1 + cx'_1 = u_1 + c(x_1 - x'_1) + cx'_1 = r_1 \\ r'_2 &= u'_2 + cx'_2 = u_2 + c(x_2 - x'_2) + cx'_2 = r_2 \end{aligned}$$

Perfect witness indistinguishability as conversations with \mathcal{P}_{x_1, x_2} and conversations with $\mathcal{P}_{x'_1, x'_2}$ are distributed identically:

$$\{(a; c; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u_1} g_2^{u_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u_1 + cx_1, u_2 + cx_2\}$$

$$\{(a; c; r_1, r_2) : u'_1, u'_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u'_1} g_2^{u'_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u'_1 + cx'_1, u'_2 + cx'_2\}$$

Witness Indistinguishability of Okamoto's Protocol

Consider conversation $(a; c; r_1, r_2)$ between \mathcal{P}_{x_1, x_2} and \mathcal{V}^* , where $a = g_1^{u_1} g_2^{u_2}$.

Let (x'_1, x'_2) be any witness satisfying $h = g_1^{x'_1} g_2^{x'_2}$.

Then unique u'_1, u'_2 **exist** that yield same conversation $(a; c; r_1, r_2)$ for $\mathcal{P}_{x'_1, x'_2}$:

$$u'_1, u'_2 \leftarrow_n u_1 + c(x_1 - x'_1), u_2 + c(x_2 - x'_2)$$

Indeed:

$$\begin{aligned} a' &= g_1^{u'_1} g_2^{u'_2} = g_1^{u_1} g_2^{u_2} (g_1^{x_1} g_2^{x_2})^c / (g_1^{x'_1} g_2^{x'_2})^c = a \\ r'_1 &= u'_1 + cx'_1 = u_1 + c(x_1 - x'_1) + cx'_1 = r_1 \\ r'_2 &= u'_2 + cx'_2 = u_2 + c(x_2 - x'_2) + cx'_2 = r_2 \end{aligned}$$

Perfect witness indistinguishability as conversations with \mathcal{P}_{x_1, x_2} and conversations with $\mathcal{P}_{x'_1, x'_2}$ are distributed identically:

$$\begin{aligned} \{(a; c; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u_1} g_2^{u_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u_1 + cx_1, u_2 + cx_2\} \\ \{(a; c; r_1, r_2) : u'_1, u'_2 \in_R \mathbb{Z}_n; a \leftarrow g_1^{u'_1} g_2^{u'_2}; c \leftarrow \mathcal{V}^*(a); r_1, r_2 \leftarrow_n u'_1 + cx'_1, u'_2 + cx'_2\} \end{aligned}$$

Witness Hidingness of Okamoto's Protocol

Proposition (Okamoto's protocol is witness hiding)

Under DL assumption, no p.p.t. verifier is able to extract a prover's private key.

Proof.

Suppose \mathcal{V}^* finds (x'_1, x'_2) by running protocol with \mathcal{P}_{x_1, x_2} .

Witness indistinguishability implies $\Pr[(x'_1, x'_2) = (x_1, x_2)] = 1/n$.

Define p.p.t. algorithm \mathcal{E} as \mathcal{P}_{x_1, x_2} and \mathcal{V}^* combined: $\mathcal{P}_{x_1, x_2} \parallel \mathcal{V}^*$



Then \mathcal{E} computes with probability $1 - \frac{1}{n}$ two pairs $(x_1, x_2) \neq (x'_1, x'_2)$ satisfying

$$h = g_1^{x_1} g_2^{x_2}, \quad h = g_1^{x'_1} g_2^{x'_2}.$$

But this implies: $g_2 = g_1^{(x'_1 - x_1)/(x'_2 - x_2)}$, hence \mathcal{E} computes $\log_{g_1} g_2$, in contradiction with DL assumption.

Witness Hidingness of Okamoto's Protocol

Proposition (Okamoto's protocol is witness hiding)

Under DL assumption, no p.p.t. verifier is able to extract a prover's private key.

Proof.

Suppose \mathcal{V}^* finds (x'_1, x'_2) by running protocol with \mathcal{P}_{x_1, x_2} .

Witness indistinguishability implies $\Pr[(x'_1, x'_2) = (x_1, x_2)] = 1/n$.

Define p.p.t. algorithm \mathcal{E} as \mathcal{P}_{x_1, x_2} and \mathcal{V}^* combined: $\mathcal{P}_{x_1, x_2} \parallel \mathcal{V}^*$



Then \mathcal{E} computes with probability $1 - \frac{1}{n}$ two pairs $(x_1, x_2) \neq (x'_1, x'_2)$ satisfying

$$h = g_1^{x_1} g_2^{x_2}, \quad h = g_1^{x'_1} g_2^{x'_2}.$$

But this implies: $g_2 = g_1^{(x'_1 - x_1)/(x'_2 - x_2)}$, hence \mathcal{E} computes $\log_{g_1} g_2$, in contradiction with DL assumption.

Witness Hidingness of Okamoto's Protocol

Proposition (Okamoto's protocol is witness hiding)

Under DL assumption, no p.p.t. verifier is able to extract a prover's private key.

Proof.

Suppose \mathcal{V}^* finds (x'_1, x'_2) by running protocol with \mathcal{P}_{x_1, x_2} .

Witness indistinguishability implies $\Pr[(x'_1, x'_2) = (x_1, x_2)] = 1/n$.

Define p.p.t. algorithm \mathcal{E} as \mathcal{P}_{x_1, x_2} and \mathcal{V}^* combined: \mathcal{P}_{x_1, x_2}



Then \mathcal{E} computes with probability $1 - \frac{1}{n}$ two pairs $(x_1, x_2) \neq (x'_1, x'_2)$ satisfying

$$h = g_1^{x_1} g_2^{x_2}, \quad h = g_1^{x'_1} g_2^{x'_2}.$$

But this implies: $g_2 = g_1^{(x'_1 - x_1)/(x'_2 - x_2)}$, hence \mathcal{E} computes $\log_{g_1} g_2$, in contradiction with DL assumption.

Witness Hidingness of Okamoto's Protocol

Proposition (Okamoto's protocol is witness hiding)

Under DL assumption, no p.p.t. verifier is able to extract a prover's private key.

Proof.

Suppose \mathcal{V}^* finds (x'_1, x'_2) by running protocol with \mathcal{P}_{x_1, x_2} .

Witness indistinguishability implies $\Pr[(x'_1, x'_2) = (x_1, x_2)] = 1/n$.

Define p.p.t. algorithm \mathcal{E} as \mathcal{P}_{x_1, x_2} and \mathcal{V}^* combined: $\mathcal{P}_{x_1, x_2} \parallel \mathcal{V}^*$



Then \mathcal{E} computes with probability $1 - \frac{1}{n}$ two pairs $(x_1, x_2) \neq (x'_1, x'_2)$ satisfying

$$h = g_1^{x_1} g_2^{x_2}, \quad h = g_1^{x'_1} g_2^{x'_2}.$$

But this implies: $g_2 = g_1^{(x'_1 - x_1)/(x'_2 - x_2)}$, hence \mathcal{E} computes $\log_{g_1} g_2$, in contradiction with DL assumption.

Witness Hidingness of Okamoto's Protocol

Proposition (Okamoto's protocol is witness hiding)

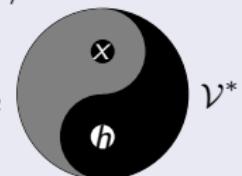
Under DL assumption, no p.p.t. verifier is able to extract a prover's private key.

Proof.

Suppose \mathcal{V}^* finds (x'_1, x'_2) by running protocol with \mathcal{P}_{x_1, x_2} .

Witness indistinguishability implies $\Pr[(x'_1, x'_2) = (x_1, x_2)] = 1/n$.

Define p.p.t. algorithm \mathcal{E} as \mathcal{P}_{x_1, x_2} and \mathcal{V}^* combined: $\mathcal{P}_{x_1, x_2} \parallel \mathcal{V}^*$



Then \mathcal{E} computes with probability $1 - \frac{1}{n}$ two pairs $(x_1, x_2) \neq (x'_1, x'_2)$ satisfying

$$h = g_1^{x_1} g_2^{x_2}, \quad h = g_1^{x'_1} g_2^{x'_2}.$$

But this implies: $g_2 = g_1^{(x'_1 - x_1)/(x'_2 - x_2)}$, hence \mathcal{E} computes $\log_{g_1} g_2$, in contradiction with DL assumption.

Witness Hidingness of Okamoto's Protocol

Proposition (Okamoto's protocol is witness hiding)

Under DL assumption, no p.p.t. verifier is able to extract a prover's private key.

Proof.

Suppose \mathcal{V}^* finds (x'_1, x'_2) by running protocol with \mathcal{P}_{x_1, x_2} .

Witness indistinguishability implies $\Pr[(x'_1, x'_2) = (x_1, x_2)] = 1/n$.

Define p.p.t. algorithm \mathcal{E} as \mathcal{P}_{x_1, x_2} and \mathcal{V}^* combined: $\mathcal{P}_{x_1, x_2} \parallel \mathcal{V}^*$

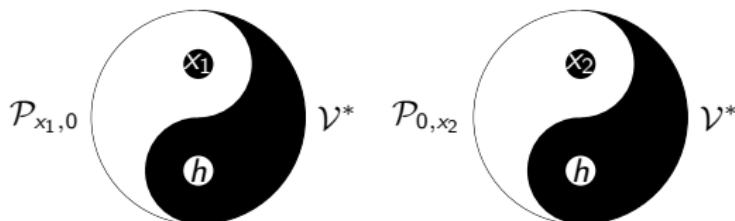


Then \mathcal{E} computes with probability $1 - \frac{1}{n}$ two pairs $(x_1, x_2) \neq (x'_1, x'_2)$ satisfying

$$h = g_1^{x_1} g_2^{x_2}, \quad h = g_1^{x'_1} g_2^{x'_2}.$$

But this implies: $g_2 = g_1^{(x'_1 - x_1)/(x'_2 - x_2)}$, hence \mathcal{E} computes $\log_{g_1} g_2$, in contradiction with DL assumption.

Witness Hidingness of Okamoto's Protocol



V^* cannot distinguish between protocol with $\mathcal{P}_{x_1,0}$ and protocol with \mathcal{P}_{0,x_2}
 \Rightarrow protocol is witness indistinguishable

Remark 4.5

There are n possible witnesses (x_1, x_2) for a given public key $h = g_1^{x_1} g_2^{x_2}$ in Okamoto's protocol. The protocol remains witness hiding even if the number of possible witnesses used by \mathcal{P} is limited to just two. For instance, if either $x_1 = 0$ or $x_2 = 0$ (hence either $h = g_1^{x_1}$ or $h = g_2^{x_2}$), the same analysis applies, except that the probability for two different witnesses is now bounded below by $1/2$.

Zero-Knowledge Proofs for NP-Statements

Schnorr identification protocol is zero-knowledge proof for statement

“I know private key x for public key $h = g^x$ ”

Corresponding relation: $R = \{(h; x) : h = g^x\} \subseteq \langle g \rangle \times \mathbb{Z}_n$

NP-statement: “I know witness w satisfying $(v; w) \in R$ ”

$R = \{(v; w)\} \subseteq V \times W$: easy to check binary relation

$v \in V$: common input to prover and verifier

$w \in W$ s.t. $(v; w) \in R$: **witness**, private input to prover

$L_R = \{v \in V : \exists_{w \in W} (v; w) \in R\}$: NP-language corresponding to relation R

Zero-Knowledge Proofs for NP-Statements

Schnorr identification protocol is zero-knowledge proof for statement

“I know private key x for public key $h = g^x$ ”

Corresponding relation: $R = \{(h; x) : h = g^x\} \subseteq \langle g \rangle \times \mathbb{Z}_n$

NP-statement: “I know witness w satisfying $(v; w) \in R$ ”

$R = \{(v; w)\} \subseteq V \times W$: easy to check binary relation

$v \in V$: common input to prover and verifier

$w \in W$ s.t. $(v; w) \in R$: **witness**, private input to prover

$L_R = \{v \in V : \exists_{w \in W} (v; w) \in R\}$: NP-language corresponding to relation R

Zero-Knowledge Proofs for NP-Statements

Schnorr identification protocol is zero-knowledge proof for statement

“I know private key x for public key $h = g^x$ ”

Corresponding relation: $R = \{(h; x) : h = g^x\} \subseteq \langle g \rangle \times \mathbb{Z}_n$

NP-statement: “I know witness w satisfying $(v; w) \in R$ ”

$R = \{(v; w)\} \subseteq V \times W$: easy to check binary relation

$v \in V$: common input to prover and verifier

$w \in W$ s.t. $(v; w) \in R$: **witness**, private input to prover

$L_R = \{v \in V : \exists_{w \in W} (v; w) \in R\}$: NP-language corresponding to relation R

Σ -Protocols

Σ -protocol **well-chosen abstraction** of many protocols incl. identification protocols by Schnorr, Guillou-Quisquater, Okamoto.

Many variations of Σ -protocols exists in today's literature.

Σ -Protocols

Σ -protocol **well-chosen abstraction** of many protocols incl. identification protocols by Schnorr, Guillou-Quisquater, Okamoto.

Many variations of Σ -protocols exists in today's literature.

Not to mention this one:

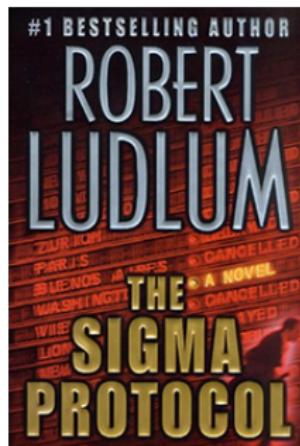
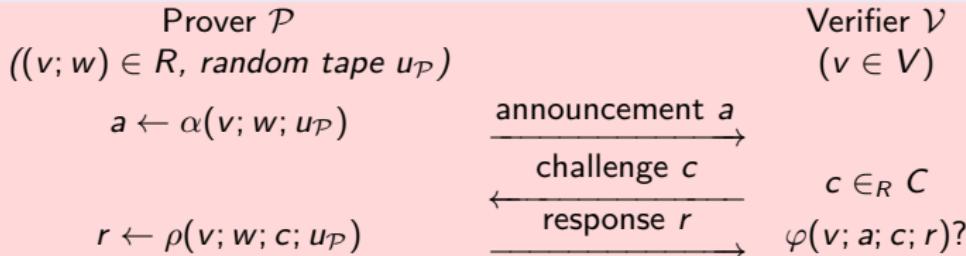


Figure 5.1 (Σ -protocol for relation R (p.p.t. algorithms α, ρ , p.t. predicate φ))

Definition 5.1

Σ -protocol for relation R is a protocol of the above form satisfying:

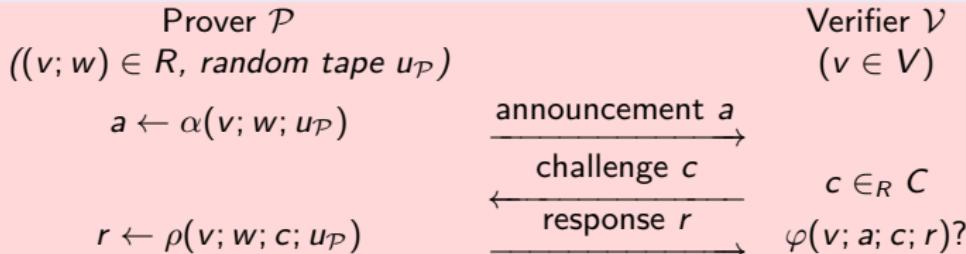
Completeness. If \mathcal{P} and \mathcal{V} follow the protocol, then \mathcal{V} always accepts.

Special soundness. Efficient extractor E : given $v \in V$ and accepting conversations $(a; c; r)$ and $(a; c'; r')$, $c \neq c'$, output witness w with $(v; w) \in R$.

Special honest-verifier zero-knowledge. Efficient simulator S : given $v \in L_R$ and challenge c , output conversations $(a; c; r)$ with same probability distribution as conversations between honest \mathcal{P} and \mathcal{V} on common input v and challenge c , where \mathcal{P} uses any witness w with $(v; w) \in R$.

For $v \in V \setminus L_R$, S outputs arbitrary accepting conversations with challenge c .

Figure 5.1 (Σ -protocol for relation R (p.p.t. algorithms α, ρ , p.t. predicate φ))



Definition 5.1

Σ -protocol for relation R is a protocol of the above form satisfying:

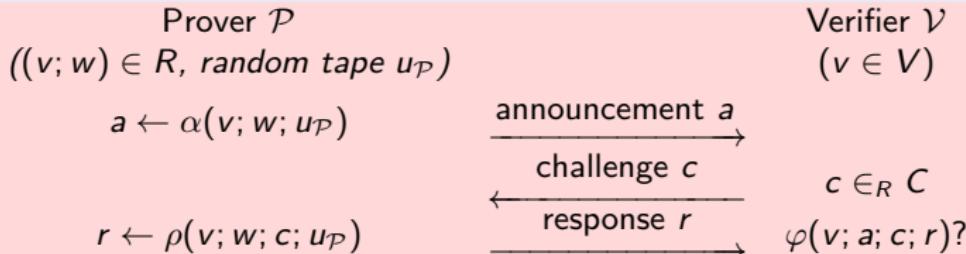
Completeness. If \mathcal{P} and \mathcal{V} follow the protocol, then \mathcal{V} always accepts.

Special soundness. Efficient extractor E : given $v \in V$ and accepting conversations $(a; c; r)$ and $(a; c'; r')$, $c \neq c'$, output witness w with $(v; w) \in R$.

Special honest-verifier zero-knowledge. Efficient simulator S : given $v \in L_R$ and challenge c , output conversations $(a; c; r)$ with same probability distribution as conversations between honest \mathcal{P} and \mathcal{V} on common input v and challenge c , where \mathcal{P} uses any witness w with $(v; w) \in R$.

For $v \in V \setminus L_R$, S outputs arbitrary accepting conversations with challenge c .

Figure 5.1 (Σ -protocol for relation R (p.p.t. algorithms α, ρ , p.t. predicate φ))



Definition 5.1

Σ -protocol for relation R is a protocol of the above form satisfying:

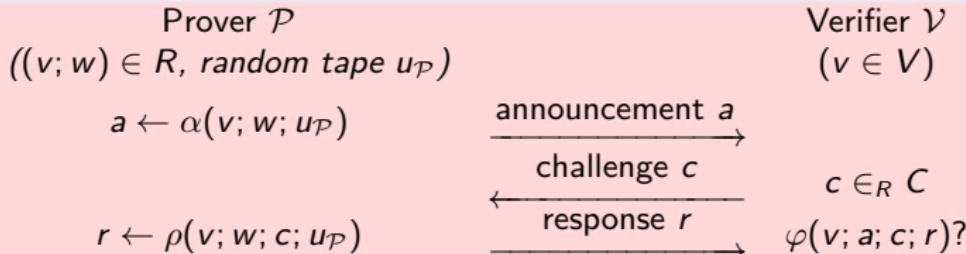
Completeness. If \mathcal{P} and \mathcal{V} follow the protocol, then \mathcal{V} always accepts.

Special soundness. Efficient extractor E : given $v \in V$ and accepting conversations $(a; c; r)$ and $(a; c'; r')$, $c \neq c'$, output witness w with $(v; w) \in R$.

Special honest-verifier zero-knowledge. Efficient simulator S : given $v \in L_R$ and challenge c , output conversations $(a; c; r)$ with same probability distribution as conversations between honest \mathcal{P} and \mathcal{V} on common input v and challenge c , where \mathcal{P} uses any witness w with $(v; w) \in R$.

For $v \in V \setminus L_R$, S outputs arbitrary accepting conversations with challenge c .

Figure 5.1 (Σ -protocol for relation R (p.p.t. algorithms $\alpha, \rho, \text{p.t. predicate } \varphi$))



Definition 5.1

Σ -protocol for relation R is a protocol of the above form satisfying:

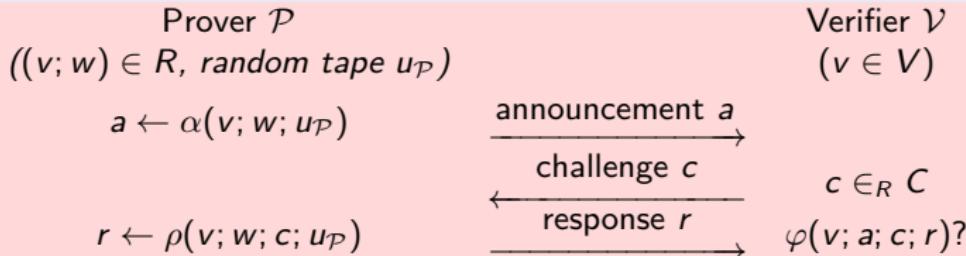
Completeness. If \mathcal{P} and \mathcal{V} follow the protocol, then \mathcal{V} always accepts.

Special soundness. Efficient extractor E : given $v \in V$ and accepting conversations $(a; c; r)$ and $(a; c'; r')$, $c \neq c'$, output witness w with $(v; w) \in R$.

Special honest-verifier zero-knowledge. Efficient simulator S : given $v \in L_R$ and challenge c , output conversations $(a; c; r)$ with same probability distribution as conversations between honest \mathcal{P} and \mathcal{V} on common input v and challenge c , where \mathcal{P} uses any witness w with $(v; w) \in R$.

For $v \in V \setminus L_R$, S outputs arbitrary accepting conversations with challenge c .

Figure 5.1 (Σ -protocol for relation R (p.p.t. algorithms α, ρ , p.t. predicate φ))



Definition 5.1

Σ -protocol for relation R is a protocol of the above form satisfying:

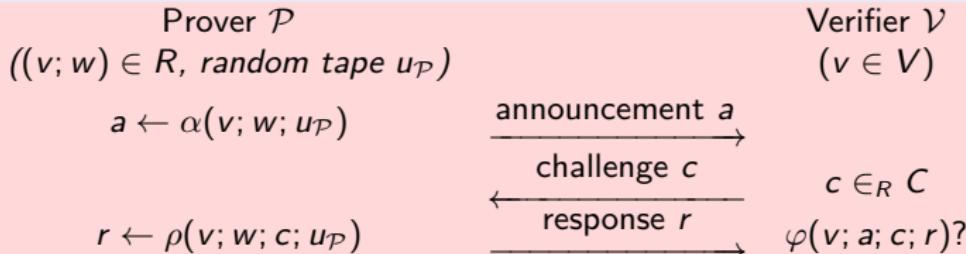
Completeness. If \mathcal{P} and \mathcal{V} follow the protocol, then \mathcal{V} always accepts.

Special soundness. Efficient **extractor** E : given $v \in V$ and accepting conversations $(a; c; r)$ and $(a; c'; r')$, $c \neq c'$, output witness w with $(v; w) \in R$.

Special honest-verifier zero-knowledge. Efficient **simulator** S : given $v \in L_R$ and challenge c , output conversations $(a; c; r)$ with same probability distribution as conversations between honest \mathcal{P} and \mathcal{V} on common input v and challenge c , where \mathcal{P} uses any witness w with $(v; w) \in R$.

For $v \in V \setminus L_R$, S outputs arbitrary accepting conversations with challenge c .

Figure 5.1 (Σ -protocol for relation R (p.p.t. algorithms α, ρ , p.t. predicate φ))



Definition 5.1

Σ -protocol for relation R is a protocol of the above form satisfying:

Completeness. If \mathcal{P} and \mathcal{V} follow the protocol, then \mathcal{V} always accepts.

Special soundness. Efficient **extractor** E : given $v \in V$ and accepting conversations $(a; c; r)$ and $(a; c'; r')$, $c \neq c'$, output witness w with $(v; w) \in R$.

Special honest-verifier zero-knowledge. Efficient **simulator** S : given $v \in L_R$ and challenge c , output conversations $(a; c; r)$ with same probability distribution as conversations between honest \mathcal{P} and \mathcal{V} on common input v and challenge c , where \mathcal{P} uses any witness w with $(v; w) \in R$.

For $v \in V \setminus L_R$, S outputs arbitrary accepting conversations with challenge c .

Plain vs Special Honest Verifier Zero-Knowledge

plain honest-verifier zero-knowledge:

- no input c for simulator S
- simulator is free to output conversation with any challenge c

special honest-verifier zero-knowledge:

- input c for simulator S
- simulator must output conversation for challenge c

special honest-verifier zero-knowledge \Rightarrow **plain** honest-verifier zero-knowledge

But special honest-verifier zero-knowledge is not much stronger.

Plain vs Special Honest Verifier Zero-Knowledge

plain honest-verifier zero-knowledge:

- **no input c for simulator S**
- simulator is free to output conversation with any challenge c

special honest-verifier zero-knowledge:

- **input c for simulator S**
- simulator must output conversation for challenge c

special honest-verifier zero-knowledge \Rightarrow **plain** honest-verifier zero-knowledge

But special honest-verifier zero-knowledge is not much stronger.

Plain vs Special Honest Verifier Zero-Knowledge

plain honest-verifier zero-knowledge:

- **no input c for simulator S**
- simulator is free to output conversation with any challenge c

special honest-verifier zero-knowledge:

- **input c for simulator S**
- simulator must output conversation for challenge c

special honest-verifier zero-knowledge \Rightarrow **plain** honest-verifier zero-knowledge

But special honest-verifier zero-knowledge is not much stronger.

Plain vs Special Honest Verifier Zero-Knowledge

plain honest-verifier zero-knowledge:

- **no input c** for simulator S
- simulator is free to output conversation with any challenge c

special honest-verifier zero-knowledge:

- **input c** for simulator S
- simulator must output conversation for challenge c

special honest-verifier zero-knowledge \Rightarrow **plain** honest-verifier zero-knowledge

But special honest-verifier zero-knowledge is not much stronger.

From Plain to Special Honest Verifier Zero-Knowledge

Without loss of generality, let $(C, +)$ be an additive finite group.

Figure 5.2 (Transformed Σ -protocol for relation R)



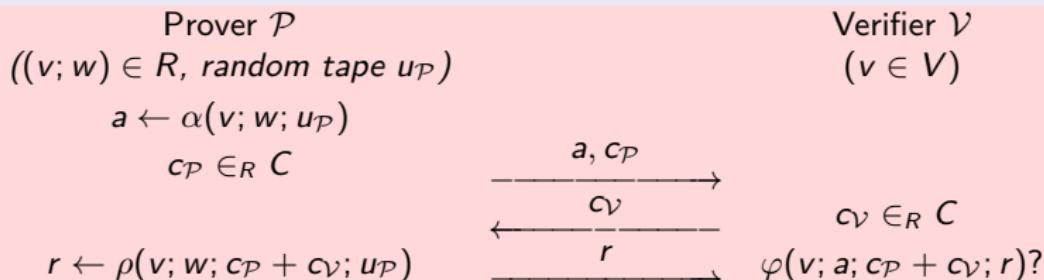
Proposition 5.2

The transformed protocol in Figure 5.2 is a Σ -protocol for relation R , provided that the original protocol as given in Figure 5.1 satisfies completeness, special soundness, and plain honest-verifier zero-knowledgeness.

From Plain to Special Honest Verifier Zero-Knowledge

Without loss of generality, let $(C, +)$ be an additive finite group.

Figure 5.2 (Transformed Σ -protocol for relation R)



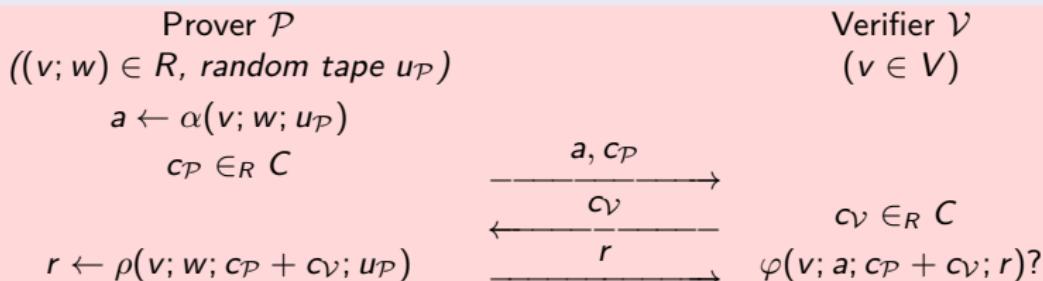
Proposition 5.2

The transformed protocol in Figure 5.2 is a Σ -protocol for relation R , provided that the original protocol as given in Figure 5.1 satisfies completeness, special soundness, and plain honest-verifier zero-knowledgeness.

From Plain to Special Honest Verifier Zero-Knowledge

Without loss of generality, let $(C, +)$ be an additive finite group.

Figure 5.2 (Transformed Σ -protocol for relation R)



Proposition 5.2

The transformed protocol in Figure 5.2 is a Σ -protocol for relation R , provided that the original protocol as given in Figure 5.1 satisfies completeness, special soundness, and plain honest-verifier zero-knowledgeness.

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. □

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. □

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. □

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. □

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. □

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. 

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. □

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$.

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. □

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$.

Proposition 5.3 (Schnorr's protocol)

The protocol in Figure 4.3 is a Σ -protocol for relation $\{(h; x) : h = g^x\}$.

Proof.

Completeness. $g^r = g^{u+cx} = g^u(g^x)^c = ah^c$.

Special soundness. For accepting $(a; c; r), (a; c'; r')$, $c \neq c'$:

$$g^r = ah^c, g^{r'} = ah^{c'} \Rightarrow g^{r-r'} = h^{c-c'} \Leftrightarrow h = g^{\frac{r-r'}{c-c'}}.$$

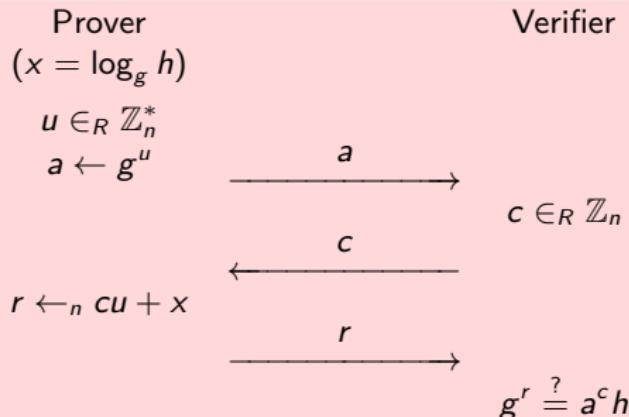
Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Special honest-verifier zero-knowledgeness. For challenge c , distributions for conversations with honest verifier and simulated conversations are resp.:

$$\begin{aligned} &\{(a; c; r) : u \in_R \mathbb{Z}_n; a \leftarrow g^u; r \leftarrow_n u + cx\}, \\ &\{(a; c; r) : r \in_R \mathbb{Z}_n; a \leftarrow g^r h^{-c}\}. \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n$. □

Figure 5.3 (Insecure variant of Schnorr's protocol)



Exercise 5.4

To see that honest-verifier zero-knowledge does not imply zero-knowledge against arbitrarily cheating verifiers, consider the protocol in Figure 5.3. Show that the protocol is complete, special sound, and honest-verifier zero-knowledge. Also, show that the protocol is completely insecure against a cheating verifier.

Exercise 5.5

The protocol in Figure 5.3 avoids the value $u = 0$, but this is not essential. Show that the protocol remains complete, special sound, and honest-verifier zero-knowledge (and insecure against a cheating verifier) if one uses $u \in_R \mathbb{Z}_n$ instead of $u \in_R \mathbb{Z}_n^$, by exhibiting a slightly more involved simulation.*

Remark 5.6

By applying the transformation of Proposition 5.2 to the protocol in Figure 5.3, we obtain a Σ -protocol which is completely insecure against a cheating verifier.

Five Forms of Composition

Parallel composition maintains Σ -protocol properties

AND-composition prove knowledge of two witnesses

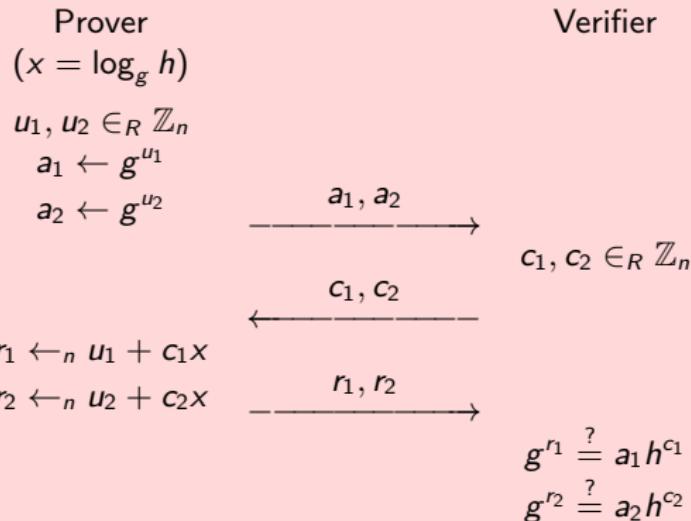
EQ-composition prove equality of witnesses

OR-composition prove knowledge of 1-out-of-2 witnesses, not revealing which

NEQ-composition prove inequality of witnesses

All illustrated for Schnorr's Σ -protocol.

Figure 5.4 (Parallel composition of Schnorr's protocol)



Proposition 5.7 (Parallel composition)

The protocol in Figure 5.4 is a Σ -protocol for relation

$$\{(h; x) : h = g^x\}$$

Proof.

Completeness.

$$\begin{aligned} g^{r_1} &= g^{u_1 + c_1 x} = g^{u_1} (g^x)^{c_1} = a_1 h^{c_1} \\ g^{r_2} &= g^{u_2 + c_2 x} = g^{u_2} (g^x)^{c_2} = a_2 h^{c_2} \end{aligned}$$

Proposition 5.7 (Parallel composition)

The protocol in Figure 5.4 is a Σ -protocol for relation

$$\{(h; x) : h = g^x\}$$

Proof.

Special soundness. Given accepting conversations $(a_1, a_2; c_1, c_2; r_1, r_2)$ and $(a_1, a_2; c'_1, c'_2; r'_1, r'_2)$ with $(c_1, c_2) \neq (c'_1, c'_2)$. Then $c_1 \neq c'_1$ and/or $c_2 \neq c'_2$.

If $c_1 \neq c'_1$:

$$g^{r_1} = a_1 h^{c_1}, g^{r'_1} = a_1 h^{c'_1} \Rightarrow g^{r_1 - r'_1} = h^{c_1 - c'_1} \Leftrightarrow h = g^{\frac{r_1 - r'_1}{c_1 - c'_1}}$$

Witness extracted as $x \leftarrow_n (r_1 - r'_1)/(c_1 - c'_1)$.

If $c_2 \neq c'_2$: same witness extracted as $x \leftarrow_n (r_2 - r'_2)/(c_2 - c'_2)$.

Hence: witness x satisfying $h = g^x$ can be extracted in either case.



Proposition 5.7 (Parallel composition)

The protocol in Figure 5.4 is a Σ -protocol for relation

$$\{(h; x) : h = g^x\}$$

Proof.

Special honest-verifier zero-knowledgeness. Given challenge (c_1, c_2) , distributions of real conversations and simulated conversations are:

$$\begin{aligned} \{(a_1, a_2; c_1, c_2; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{u_1}; a_2 \leftarrow g^{u_2}; \\ r_1 \leftarrow_n u_1 + c_1 x; r_2 \leftarrow_n u_2 + c_2 x\} \end{aligned}$$

$$\{(a_1, a_2; c_1, c_2; r_1, r_2) : r_1, r_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{r_1} h^{-c_1}; a_2 \leftarrow g^{r_2} h^{-c_2}\}$$

Identical distributions: each conversation occurs with probability $1/n^2$.



AND-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

$$R_1 \wedge R_2 := \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1, (v_2; w_2) \in R_2\}$$

So, run Σ -protocols for relations R_1 and R_2 in parallel?

Fails! See Exercise 5.9.

Run Σ -protocols for relations R_1 and R_2 in parallel, using **common challenge** c.

AND-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

$$R_1 \wedge R_2 := \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1, (v_2; w_2) \in R_2\}$$

So, run Σ -protocols for relations R_1 and R_2 in parallel?

Fails! See Exercise 5.9.

Run Σ -protocols for relations R_1 and R_2 in parallel, using **common challenge** c.



AND-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

$$R_1 \wedge R_2 := \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1, (v_2; w_2) \in R_2\}$$

So, run Σ -protocols for relations R_1 and R_2 in parallel?

Fails! See Exercise 5.9.

Run Σ -protocols for relations R_1 and R_2 in parallel, using **common challenge** c.

AND-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

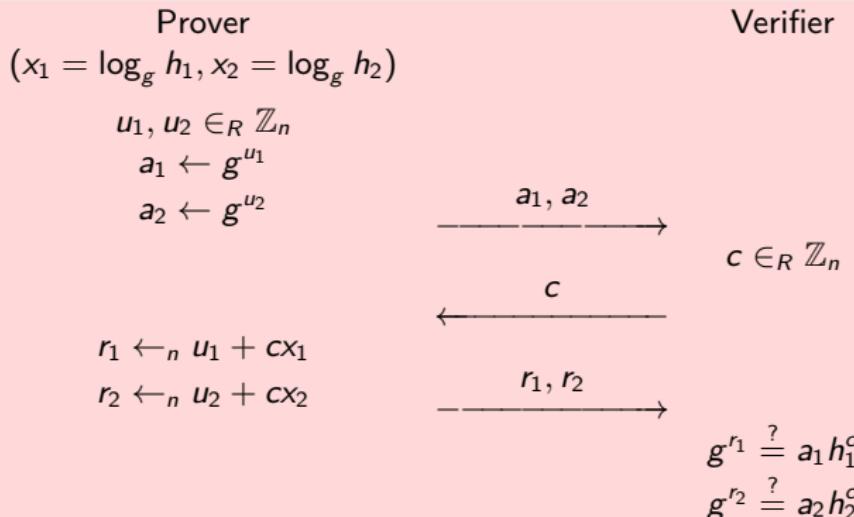
$$R_1 \wedge R_2 := \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1, (v_2; w_2) \in R_2\}$$

So, run Σ -protocols for relations R_1 and R_2 in parallel?

Fails! See Exercise 5.9.

Run Σ -protocols for relations R_1 and R_2 in parallel, using **common challenge c**.

Figure 5.5 (AND-composition of Schnorr's protocol)



Proposition 5.8 (AND-composition)

The protocol in Figure 5.5 is a Σ -protocol for relation

$$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1}, h_2 = g^{x_2}\}.$$

Proof.

Completeness.

$$\begin{aligned} g^{r_1} &= g^{u_1 + cx_1} = g^{u_1}(g^{x_1})^c = a_1 h_1^c, \\ g^{r_2} &= g^{u_2 + cx_2} = g^{u_2}(g^{x_2})^c = a_2 h_2^c. \end{aligned}$$

Proposition 5.8 (AND-composition)

The protocol in Figure 5.5 is a Σ -protocol for relation

$$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1}, h_2 = g^{x_2}\}.$$

Proof.

Special soundness. For accepting $(a_1, a_2; c; r_1, r_2), (a_1, a_2; c'; r'_1, r'_2)$, $c \neq c'$:

$$g^{r_1} = a_1 h_1^c, g^{r'_1} = a_1 h_1^{c'} \Rightarrow g^{r_1 - r'_1} = h_1^{c - c'} \Leftrightarrow h_1 = g^{\frac{r_1 - r'_1}{c - c'}}.$$

By symmetry: $h_2 = g^{\frac{r_2 - r'_2}{c - c'}}.$

Witness (x_1, x_2) extracted as $x_1 \leftarrow_n (r_1 - r'_1)/(c - c')$, $x_2 \leftarrow_n (r_2 - r'_2)/(c - c')$.



Proposition 5.8 (AND-composition)

The protocol in Figure 5.5 is a Σ -protocol for relation

$$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1}, h_2 = g^{x_2}\}.$$

Proof.

Special honest-verifier zero-knowledgeness. Given challenge c , distribution of real conversations and simulated conversations are:

$$\{(a_1, a_2; c; r_1, r_2) : u_1, u_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{u_1}; a_2 \leftarrow g^{u_2}; r_1 \leftarrow_n u_1 + cx_1; r_2 \leftarrow_n u_2 + cx_2\}$$

$$\{(a_1, a_2; c; r_1, r_2) : r_1, r_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{r_1} h_1^{-c}; a_2 \leftarrow g^{r_2} h_2^{-c}\}$$

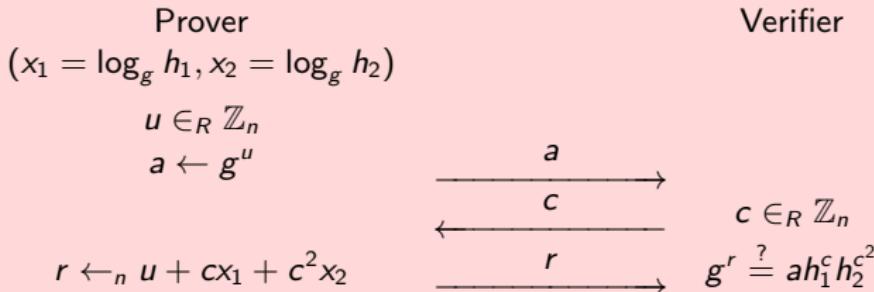
Identical distributions: each conversation occurs with probability $1/n^2$.

Exercise

Exercise 5.9

By considering the special soundness property, explain why running a Schnorr protocol for h_1 and a Schnorr protocol for h_2 in parallel does not yield a Σ -protocol for relation $\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1}, h_2 = g^{x_2}\}$.

Figure 5.6 (Alternative to AND-composition of Schnorr's protocol?)



Exercise 5.10

Consider the protocol in Figure 5.6 for relation

$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1}, h_2 = g^{x_2}\}$. (i) Show that the protocol is complete and special honest-verifier zero-knowledge. (ii) Why does special soundness not hold for this protocol? Hint: consider prover who knows $x_1 = \log_g h_1$ but does not know $x_2 = \log_g h_2$. (iii) Show that soundness holds in this sense: for any $(h_1, h_2) \in \langle g \rangle \times \langle g \rangle$, given three accepting conversations $(a; c; r)$, $(a; c'; r')$, $(a; c''; r'')$ with $c \neq c'$, $c \neq c''$, $c' \neq c''$ one can efficiently compute witness (x_1, x_2) satisfying $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$.

EQ-Composition

Given Σ -protocol for relation R , construct Σ -protocol for relation:

$$\{(v_1, v_2; w) : (v_1; w) \in R, (v_2; w) \in R\}$$

Special case of AND-composition, with common witness w for v_1 and v_2 .

Use AND-composition, but prover uses same random tape u_P (see Figure 5.1) in both cases.

Run two instances of Σ -protocol for R in parallel, using **common randomness** u_P , **common challenge** c and **common response** r .

EQ-Composition

Given Σ -protocol for relation R , construct Σ -protocol for relation:

$$\{(v_1, v_2; w) : (v_1; w) \in R, (v_2; w) \in R\}$$

Special case of AND-composition, with common witness w for v_1 and v_2 .

Use AND-composition, but prover uses **same** random tape u_P (see Figure 5.1) in both cases.

Run two instances of Σ -protocol for R in parallel, using **common randomness** u_P , **common challenge** c and **common response** r .

EQ-Composition

Given Σ -protocol for relation R , construct Σ -protocol for relation:

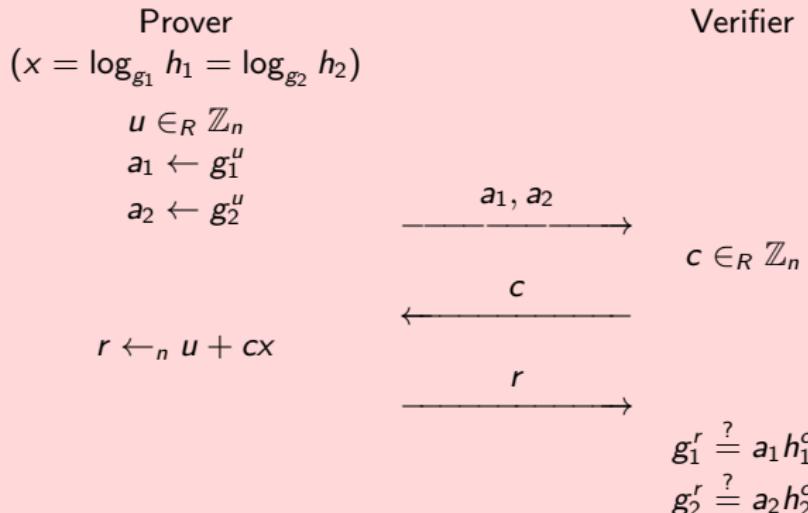
$$\{(v_1, v_2; w) : (v_1; w) \in R, (v_2; w) \in R\}$$

Special case of AND-composition, with common witness w for v_1 and v_2 .

Use AND-composition, but prover uses **same** random tape u_P (see Figure 5.1) in both cases.

Run two instances of Σ -protocol for R in parallel, using **common randomness** u_P , **common challenge** c and **common response** r .

Figure 5.7 (EQ-composition of Schnorr's protocol)



Proposition 5.11 (EQ-composition)

The protocol in Figure 5.7 is a Σ -protocol for relation

$$\{(g_1, h_1, g_2, h_2; x) : h_1 = g_1^x, h_2 = g_2^x\}.$$

Proof.

Completeness.

$$\begin{aligned} g_1^r &= g_1^{u+cx} = g_1^u(g_1^x)^c = a_1 h_1^c, \\ g_2^r &= g_2^{u+cx} = g_2^u(g_2^x)^c = a_2 h_2^c. \end{aligned}$$

Proposition 5.11 (EQ-composition)

The protocol in Figure 5.7 is a Σ -protocol for relation

$$\{(g_1, h_1, g_2, h_2; x) : h_1 = g_1^x, h_2 = g_2^x\}.$$

Proof.

Special soundness. Given accepting $(a_1, a_2; c; r)$ and $(a_1, a_2; c'; r')$, $c \neq c'$:

$$\begin{aligned} g_1^r &= a_1 h_1^c, & g_2^r &= a_2 h_2^c, & g_1^{r'} &= a_1 h_1^{c'}, & g_2^{r'} &= a_2 h_2^{c'} \\ \Rightarrow g_1^{r-r'} &= h_1^{c-c'}, & g_2^{r-r'} &= h_2^{c-c'} \\ \Leftrightarrow h_1 &= g_1^{\frac{r-r'}{c-c'}}, & h_2 &= g_2^{\frac{r-r'}{c-c'}}. \end{aligned}$$

Witness extracted as $x \leftarrow_n (r - r')/(c - c')$.

Proposition 5.11 (EQ-composition)

The protocol in Figure 5.7 is a Σ -protocol for relation

$$\{(g_1, h_1, g_2, h_2; x) : h_1 = g_1^x, h_2 = g_2^x\}.$$

Proof.

Special honest-verifier zero-knowledgeness. Given challenge c , distribution of real conversations and simulated conversations are:

$$\begin{aligned} &\{(a_1, a_2; c; r) : u \in_R \mathbb{Z}_n; a_1 \leftarrow g_1^u; a_2 \leftarrow g_2^u; r \leftarrow_n u + cx\} \\ &\{(a_1, a_2; c; r) : r \in_R \mathbb{Z}_n; a_1 \leftarrow g_1^r h_1^{-c}; a_2 \leftarrow g_2^r h_2^{-c}\} \end{aligned}$$

Identical distributions provided $\log_{g_1} h_1 = \log_{g_2} h_2$, cf. Definition 5.1; furthermore, if $\log_{g_1} h_1 \neq \log_{g_2} h_2$, simulated conversations are accepting, as required.



OR-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

$$R_1 \vee R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \vee (v_2; w_2) \in R_2\}$$

Suppose prover knows w_1 with $(v_1; w_1) \in R_1$ (but no w_2 with $(v_2; w_2) \in R_2$).

Prover can run Σ -protocol for R_1 but not for R_2 .

Verifier lets prover “cheat” by allowing prover to use **simulation** for R_2 .

Prover allowed to split challenge c into “challenges” c_1, c_2 satisfying $c_1 + c_2 = c$.

OR-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

$$R_1 \vee R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \vee (v_2; w_2) \in R_2\}$$

Suppose prover knows w_1 with $(v_1; w_1) \in R_1$ (but no w_2 with $(v_2; w_2) \in R_2$).

Prover can run Σ -protocol for R_1 but not for R_2 .

Verifier lets prover “cheat” by allowing prover to use **simulation** for R_2 .

Prover allowed to split challenge c into “challenges” c_1, c_2 satisfying $c_1 + c_2 = c$.

OR-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

$$R_1 \vee R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \vee (v_2; w_2) \in R_2\}$$

Suppose prover knows w_1 with $(v_1; w_1) \in R_1$ (but no w_2 with $(v_2; w_2) \in R_2$).

Prover can run Σ -protocol for R_1 but not for R_2 .

Verifier lets prover “cheat” by allowing prover to use **simulation** for R_2 .

Prover allowed to split challenge c into “challenges” c_1, c_2 satisfying $c_1 + c_2 = c$.

OR-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

$$R_1 \vee R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \vee (v_2; w_2) \in R_2\}$$

Suppose prover knows w_1 with $(v_1; w_1) \in R_1$ (but no w_2 with $(v_2; w_2) \in R_2$).

Prover can run Σ -protocol for R_1 but not for R_2 .

Verifier lets prover “cheat” by allowing prover to use **simulation** for R_2 .

Prover allowed to split challenge c into “challenges” c_1, c_2 satisfying $c_1 + c_2 = c$.

OR-Composition

Given Σ -protocols for relations R_1 and R_2 , construct Σ -protocol for relation:

$$R_1 \vee R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \vee (v_2; w_2) \in R_2\}$$

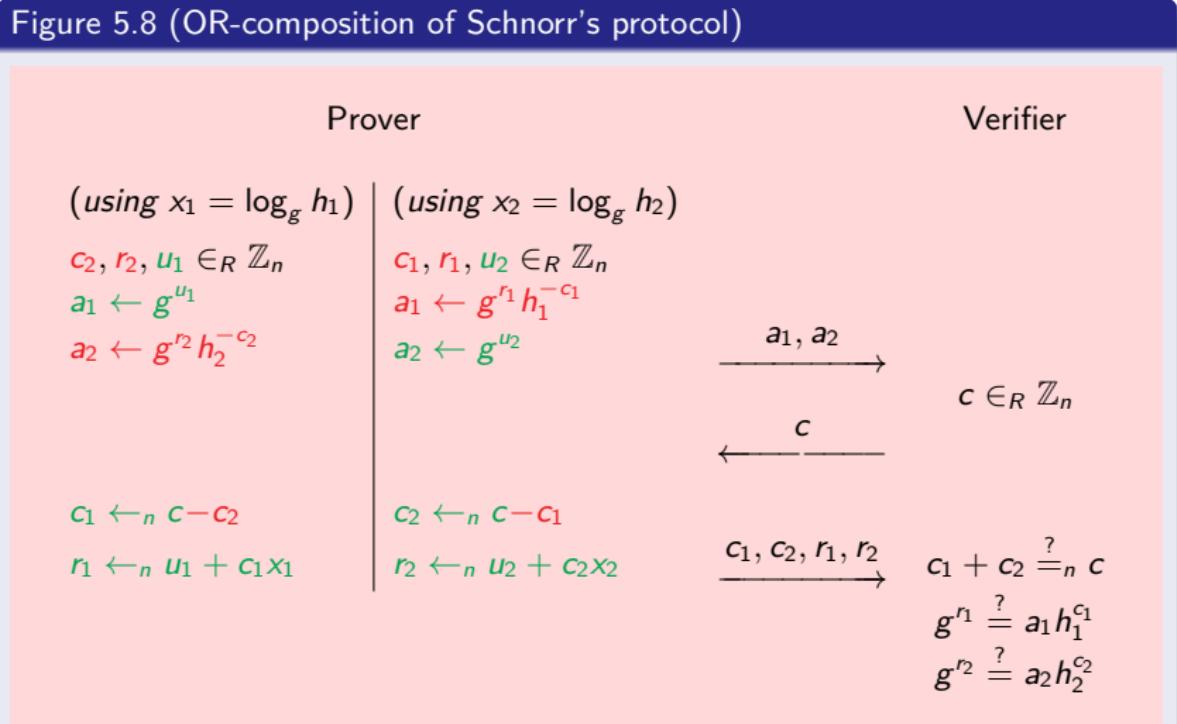
Suppose prover knows w_1 with $(v_1; w_1) \in R_1$ (but no w_2 with $(v_2; w_2) \in R_2$).

Prover can run Σ -protocol for R_1 but not for R_2 .

Verifier lets prover “cheat” by allowing prover to use **simulation** for R_2 .

Prover allowed to split challenge c into “challenges” c_1, c_2 satisfying $c_1 + c_2 = c$.

Figure 5.8 (OR-composition of Schnorr's protocol)



Proposition 5.12 (OR-composition)

The protocol in Figure 5.8 is a Σ -protocol for relation

$$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1} \vee h_2 = g^{x_2}\}.$$

Proof.

Completeness. Case prover uses x_1 . Then:

$$c_1 + c_2 = c - c_2 + c_2 = c$$

$$\begin{aligned} g^{r_1} &= g^{u_1+c_1x_1} = g^{u_1}(g^{x_1})^{c_1} = a_1 h_1^{c_1}, \\ g^{r_2} &= a_2 h_2^{c_2}. \end{aligned}$$

Case prover uses x_2 is similar.

Proposition 5.12 (OR-composition)

The protocol in Figure 5.8 is a Σ -protocol for relation

$$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1} \vee h_2 = g^{x_2}\}.$$

Proof.

Special soundness. Given accepting conversations $(a_1, a_2; c; c_1, c_2, r_1, r_2)$ and $(a_1, a_2; c'; c'_1, c'_2, r'_1, r'_2)$, $c \neq c'$. Then $c = c_1 + c_2 \neq c'_1 + c'_2 = c'$ implies $c_1 \neq c'_1$ and/or $c_2 \neq c'_2$.

$$\begin{aligned} g^{r_1} &= a_1 h_1^{c_1}, & g^{r_2} &= a_2 h_2^{c_2}, & g^{r'_1} &= a_1 h_1^{c'_1}, & g^{r'_2} &= a_2 h_2^{c'_2} \\ \Rightarrow g^{r_1 - r'_1} &= h_1^{c_1 - c'_1}, & g^{r_2 - r'_2} &= h_2^{c_2 - c'_2} \end{aligned}$$

If $c_1 \neq c'_1$, set $x_1 \leftarrow_n (r_1 - r'_1)/(c_1 - c'_1)$;
 otherwise $c_2 \neq c'_2$, and set $x_2 \leftarrow_n (r_2 - r'_2)/(c_2 - c'_2)$.
 Witness (x_1, x_2) satisfies $h_1 = g^{x_1} \vee h_2 = g^{x_2}$.

Proposition 5.12 (OR-composition)

The protocol in Figure 5.8 is a Σ -protocol for relation

$$\{(h_1, h_2; x_1, x_2) : h_1 = g^{x_1} \vee h_2 = g^{x_2}\}.$$

Proof.

Special honest-verifier zero-knowledgeness. Given challenge c , distributions of real conversations and simulated conversations are resp. (case prover uses x_1):

$$\begin{aligned} &\{(a_1, a_2; c; c_1, c_2, r_1, r_2) : u_1, c_2, r_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{u_1}; a_2 \leftarrow g^{r_2} h_2^{-c_2}; \\ &\quad c_1 \leftarrow_n c - c_2; r_1 \leftarrow_n u_1 + c_1 x_1\} \end{aligned}$$

$$\begin{aligned} &\{(a_1, a_2; c; c_1, c_2, r_1, r_2) : c_1, r_1, r_2 \in_R \mathbb{Z}_n; c_2 \leftarrow_n c - c_1; \\ &\quad a_1 \leftarrow g^{r_1} h_1^{-c_1}; a_2 \leftarrow g^{r_2} h_2^{-c_2}\} \end{aligned}$$

Identical distributions: each conversation occurs with probability $1/n^3$.

Remark (Witness indistinguishability of OR-composition)

Real conversations in case prover uses x_1 :

$$\{(a_1, a_2; c; c_1, c_2, r_1, r_2) : u_1, c_2, r_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{u_1}; a_2 \leftarrow g^{r_2} h_2^{-c_2}; c_1 \leftarrow_n c - c_2; r_1 \leftarrow_n u_1 + c_1 x_1\}$$

Real conversations in case prover uses x_2 :

$$\{(a_1, a_2; c; c_1, c_2, r_1, r_2) : u_2, c_1, r_1 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{r_1} h_1^{-c_1}; a_2 \leftarrow g^{u_2}; c_2 \leftarrow_n c - c_1; r_2 \leftarrow_n u_2 + c_2 x_2\}$$

*Identical distributions (both identical to distribution of simulated conversations). Therefore, protocol is **witness indistinguishable**.*

Exercise 5.13

As slight optimization of protocol of Figure 5.8, prover may omit sending c_2 , in which case the verifier must replace test $c_1 + c_2 \stackrel{?}{=} c$ by assignment $c_2 \leftarrow_n c - c_1$. (Thus, prover omits sending c_2 independent of whether it knows x_1 and/or x_2 .) Prove that the resulting protocol is a Σ -protocol for the same relation as before.

Remark (Witness indistinguishability of OR-composition)

Real conversations in case prover uses x_1 :

$$\{(a_1, a_2; c; c_1, c_2, r_1, r_2) : u_1, c_2, r_2 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{u_1}; a_2 \leftarrow g^{r_2} h_2^{-c_2}; c_1 \leftarrow_n c - c_2; r_1 \leftarrow_n u_1 + c_1 x_1\}$$

Real conversations in case prover uses x_2 :

$$\{(a_1, a_2; c; c_1, c_2, r_1, r_2) : u_2, c_1, r_1 \in_R \mathbb{Z}_n; a_1 \leftarrow g^{r_1} h_1^{-c_1}; a_2 \leftarrow g^{u_2}; c_2 \leftarrow_n c - c_1; r_2 \leftarrow_n u_2 + c_2 x_2\}$$

*Identical distributions (both identical to distribution of simulated conversations). Therefore, protocol is **witness indistinguishable**.*

Exercise 5.13

As slight optimization of protocol of Figure 5.8, prover may omit sending c_2 , in which case the verifier must replace test $c_1 + c_2 \stackrel{?}{=} c$ by assignment $c_2 \leftarrow_n c - c_1$. (Thus, prover omits sending c_2 independent of whether it knows x_1 and/or x_2 .) Prove that the resulting protocol is a Σ -protocol for the same relation as before.

NEQ-Composition

Given Σ -protocol for relation R , construct Σ -protocol for relation:

$$\{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R, (v_2; w_2) \in R, w_1 \neq w_2\}$$

AND-composition proves knowledge of w_1 and w_2 . How to prove $w_1 \neq w_2$?

For Schnorr's protocol, with public keys g_1, h_1 and g_2, h_2 ,
 use AND-composition to prove knowledge of $x_1 = \log_{g_1} h_1$ and $x_2 = \log_{g_2} h_2$.

Since $x_1 \neq x_2$, multiplicative inverse of $x_1 - x_2$ modulo n is defined. Hence,

$$h_1 h_2 = g_1^{x_1} g_2^{x_2} = (g_1 g_2)^{x_1} g_2^{x_2 - x_1}$$

implies

$$g_2 = (g_1 g_2)^{x_1/(x_1 - x_2)} (h_1 h_2)^{1/(x_2 - x_1)}. \quad (5.1)$$

Use instance of Okamoto's protocol with public key g_2 , generators $g_1 g_2$ and $h_1 h_2$, witnesses $x_1/(x_1 - x_2)$ and $1/(x_2 - x_1)$.

NEQ-Composition

Given Σ -protocol for relation R , construct Σ -protocol for relation:

$$\{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R, (v_2; w_2) \in R, w_1 \neq w_2\}$$

AND-composition proves knowledge of w_1 and w_2 . How to prove $w_1 \neq w_2$?

For Schnorr's protocol, with public keys g_1, h_1 and g_2, h_2 ,
 use AND-composition to prove knowledge of $x_1 = \log_{g_1} h_1$ and $x_2 = \log_{g_2} h_2$.

Since $x_1 \neq x_2$, multiplicative inverse of $x_1 - x_2$ modulo n is defined. Hence,

$$h_1 h_2 = g_1^{x_1} g_2^{x_2} = (g_1 g_2)^{x_1} g_2^{x_2 - x_1}$$

implies

$$g_2 = (g_1 g_2)^{x_1/(x_1 - x_2)} (h_1 h_2)^{1/(x_2 - x_1)}. \quad (5.1)$$

Use instance of Okamoto's protocol with public key g_2 , generators $g_1 g_2$ and $h_1 h_2$, witnesses $x_1/(x_1 - x_2)$ and $1/(x_2 - x_1)$.

NEQ-Composition

Given Σ -protocol for relation R , construct Σ -protocol for relation:

$$\{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R, (v_2; w_2) \in R, w_1 \neq w_2\}$$

AND-composition proves knowledge of w_1 and w_2 . How to prove $w_1 \neq w_2$?

For Schnorr's protocol, with public keys g_1, h_1 and g_2, h_2 ,
 use AND-composition to prove knowledge of $x_1 = \log_{g_1} h_1$ and $x_2 = \log_{g_2} h_2$.

Since $x_1 \neq x_2$, multiplicative inverse of $x_1 - x_2$ modulo n is defined. Hence,

$$h_1 h_2 = g_1^{x_1} g_2^{x_2} = (g_1 g_2)^{x_1} g_2^{x_2 - x_1}$$

implies

$$g_2 = (g_1 g_2)^{x_1/(x_1 - x_2)} (h_1 h_2)^{1/(x_2 - x_1)}. \quad (5.1)$$

Use instance of Okamoto's protocol with public key g_2 , generators $g_1 g_2$ and $h_1 h_2$, witnesses $x_1/(x_1 - x_2)$ and $1/(x_2 - x_1)$.

NEQ-Composition

Given Σ -protocol for relation R , construct Σ -protocol for relation:

$$\{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R, (v_2; w_2) \in R, w_1 \neq w_2\}$$

AND-composition proves knowledge of w_1 and w_2 . How to prove $w_1 \neq w_2$?

For Schnorr's protocol, with public keys g_1, h_1 and g_2, h_2 ,
 use AND-composition to prove knowledge of $x_1 = \log_{g_1} h_1$ and $x_2 = \log_{g_2} h_2$.

Since $x_1 \neq x_2$, multiplicative inverse of $x_1 - x_2$ modulo n is defined. Hence,

$$h_1 h_2 = g_1^{x_1} g_2^{x_2} = (g_1 g_2)^{x_1} g_2^{x_2 - x_1}$$

implies

$$g_2 = (g_1 g_2)^{x_1/(x_1 - x_2)} (h_1 h_2)^{1/(x_2 - x_1)}. \quad (5.1)$$

Use instance of Okamoto's protocol with public key g_2 , generators $g_1 g_2$ and $h_1 h_2$, witnesses $x_1/(x_1 - x_2)$ and $1/(x_2 - x_1)$.

NEQ-Composition

Given Σ -protocol for relation R , construct Σ -protocol for relation:

$$\{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R, (v_2; w_2) \in R, w_1 \neq w_2\}$$

AND-composition proves knowledge of w_1 and w_2 . How to prove $w_1 \neq w_2$?

For Schnorr's protocol, with public keys g_1, h_1 and g_2, h_2 ,
 use AND-composition to prove knowledge of $x_1 = \log_{g_1} h_1$ and $x_2 = \log_{g_2} h_2$.

Since $x_1 \neq x_2$, multiplicative inverse of $x_1 - x_2$ modulo n is defined. Hence,

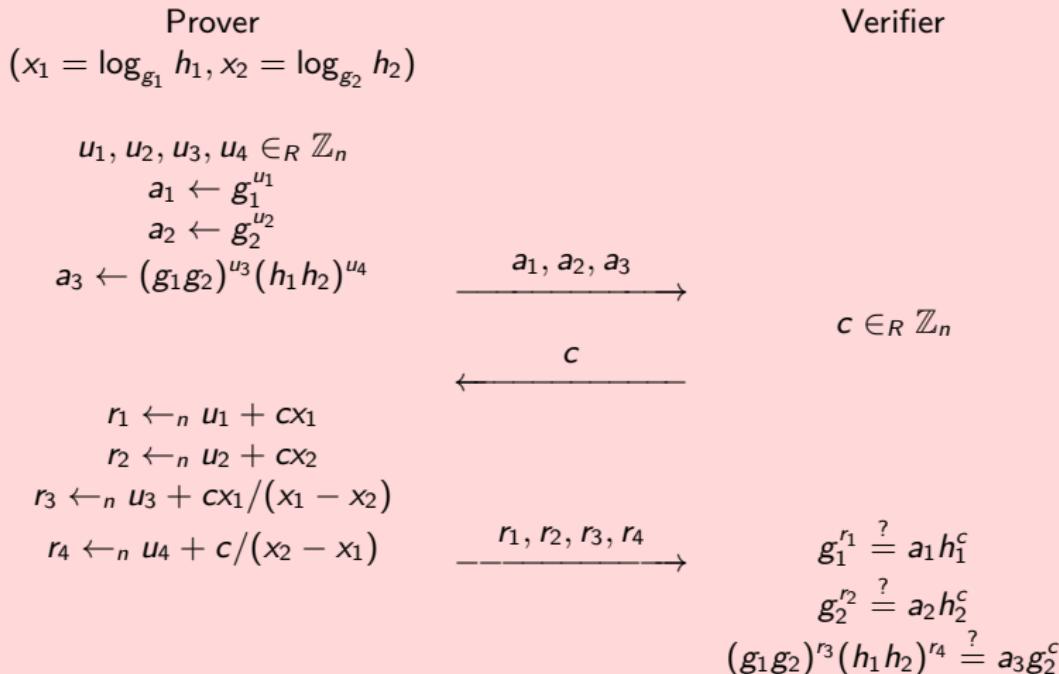
$$h_1 h_2 = g_1^{x_1} g_2^{x_2} = (g_1 g_2)^{x_1} g_2^{x_2 - x_1}$$

implies

$$g_2 = (g_1 g_2)^{x_1/(x_1 - x_2)} (h_1 h_2)^{1/(x_2 - x_1)}. \quad (5.1)$$

Use instance of Okamoto's protocol with public key g_2 , generators $g_1 g_2$ and $h_1 h_2$, witnesses $x_1/(x_1 - x_2)$ and $1/(x_2 - x_1)$.

Figure 5.9 (NEQ-composition of Schnorr's protocol)



Proposition 5.14 (NEQ-composition)

The protocol in Figure 5.9 is a Σ -protocol for relation

$$\{(g_1, h_1, g_2, h_2; x_1, x_2) : h_1 = g_1^{x_1}, h_2 = g_2^{x_2}, x_1 \neq x_2\},$$

assuming $\log_{g_1} g_2$ is unknown.

Proof.

Completeness. Clearly, $g_i^{r_i} = a_i h_i^c$ for $i = 1, 2$.

Furthermore:

$$(g_1 g_2)^{r_3} (h_1 h_2)^{r_4} = (g_1 g_2)^{u_3} (h_1 h_2)^{u_4} ((g_1 g_2)^{cx_1/(x_1 - x_2)} (h_1 h_2)^{c/(x_2 - x_1)}) = a_3 g_2^c,$$

using Eq. (5.1).

Proposition 5.14 (NEQ-composition)

The protocol in Figure 5.9 is a Σ -protocol for relation

$$\{(g_1, h_1, g_2, h_2; x_1, x_2) : h_1 = g_1^{x_1}, h_2 = g_2^{x_2}, x_1 \neq x_2\},$$

assuming $\log_{g_1} g_2$ is unknown.

Proof.

Special soundness. Given $(a_1, a_2, a_3; c; r_1, r_2, r_3, r_4), (a_1, a_2, a_3; c'; r'_1, r'_2, r'_3, r'_4)$ both accepting, $c \neq c'$. As for AND-composition, witness (x_1, x_2) is extracted as $x_1 \leftarrow_n (r'_1 - r_1)/(c' - c)$ and $x_2 \leftarrow_n (r'_2 - r_2)/(c' - c)$.

Moreover,

$$g_2 = (g_1 g_2)^{(r_3 - r'_3)/(c - c')} (h_1 h_2)^{(r_4 - r'_4)/(c - c')}.$$

Suppose $x_1 = x_2$. Then $g_2 = (g_1 g_2)^\alpha$ for known $\alpha \neq 0, 1$ (as $g_1, g_2 \neq 1$), hence $g_2 = g_1^{\alpha/(1-\alpha)}$, contradicting that $\log_{g_1} g_2$ is unknown. Therefore $x_1 \neq x_2$.

Proposition 5.14 (NEQ-composition)

The protocol in Figure 5.9 is a Σ -protocol for relation

$$\{(g_1, h_1, g_2, h_2; x_1, x_2) : h_1 = g_1^{x_1}, h_2 = g_2^{x_2}, x_1 \neq x_2\},$$

assuming $\log_{g_1} g_2$ is unknown.

Proof.

Special honest-verifier zero-knowledgeness. Given challenge c , distributions of real conversations and simulated conversations are:

$$\begin{aligned} &\{(a_1, a_2, a_3; c; r_1, r_2, r_3, r_4) : u_1, u_2, u_3, u_4 \in_R \mathbb{Z}_n; a_1 \leftarrow g_1^{u_1}; a_2 \leftarrow g_2^{u_2}; \\ &a_3 \leftarrow (g_1 g_2)^{u_3} (h_1 h_2)^{u_4}; r_1 \leftarrow_n u_1 + cx_1; r_2 \leftarrow_n u_2 + cx_2; \\ &r_3 \leftarrow_n u_3 + cx_1/(x_1 - x_2); r_4 \leftarrow_n u_4 + c/(x_2 - x_1)\}, \end{aligned}$$

$$\begin{aligned} &\{(a_1, a_2, a_3; c; r_1, r_2, r_3, r_4) : r_1, r_2, r_3, r_4 \in_R \mathbb{Z}_n; a_1 \leftarrow g_1^{r_1} h_1^{-c}; a_2 \leftarrow g_2^{r_2} h_2^{-c}; \\ &a_3 \leftarrow (g_1 g_2)^{r_3} (h_1 h_2)^{r_4} g_2^{-c}\}. \end{aligned}$$

Identical distributions provided $\log_{g_1} h_1 \neq \log_{g_2} h_2$, cf. Definition 5.1; furthermore, if $\log_{g_1} h_1 = \log_{g_2} h_2$, simulated conversations are accepting, as required.



Example 5.15

Σ -protocol needed for relation R :

$$R = \{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}.$$

Distinguish cases $x = 0$ and $x = 1$ for given $(A, B; x, y, z) \in R$.

If $x = 0$, then $A = h^y \wedge B = h^z$; if $x = 1$, then $A = gh^y \wedge B = g^y$.

For relation R , apply OR-composition to:

$$R_0 = \{(A, B; y, z) : A = h^y \wedge B = h^z\},$$

$$R_1 = \{(A, B; y) : A = gh^y \wedge B = g^y\}.$$

For relation R_0 , apply AND-composition to:

$$R_{0A} = \{(A; y) : A = h^y\}, \quad R_{0B} = \{(B; z) : B = h^z\}.$$

For relation R_1 , apply EQ-composition to:

$$R_{1A} = \{(A; y) : A/g = h^y\}, \quad R_{1B} = \{(B; y) : B = g^y\}.$$

$R_{0A}, R_{0B}, R_{1A}, R_{1B}$ are instances of (slight variations of) Schnorr's protocol.

Example 5.15

Σ -protocol needed for relation R :

$$R = \{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}.$$

Distinguish cases $x = 0$ and $x = 1$ for given $(A, B; x, y, z) \in R$.

If $x = 0$, then $A = h^y \wedge B = h^z$; if $x = 1$, then $A = gh^y \wedge B = g^y$.

For relation R , apply OR-composition to:

$$R_0 = \{(A, B; y, z) : A = h^y \wedge B = h^z\},$$

$$R_1 = \{(A, B; y) : A = gh^y \wedge B = g^y\}.$$

For relation R_0 , apply AND-composition to:

$$R_{0A} = \{(A; y) : A = h^y\}, \quad R_{0B} = \{(B; z) : B = h^z\}.$$

For relation R_1 , apply EQ-composition to:

$$R_{1A} = \{(A; y) : A/g = h^y\}, \quad R_{1B} = \{(B; y) : B = g^y\}.$$

$R_{0A}, R_{0B}, R_{1A}, R_{1B}$ are instances of (slight variations of) Schnorr's protocol.

Example 5.15

Σ -protocol needed for relation R :

$$R = \{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}.$$

Distinguish cases $x = 0$ and $x = 1$ for given $(A, B; x, y, z) \in R$.

If $x = 0$, then $A = h^y \wedge B = h^z$; if $x = 1$, then $A = gh^y \wedge B = g^y$.

For relation R , apply OR-composition to:

$$R_0 = \{(A, B; y, z) : A = h^y \wedge B = h^z\},$$

$$R_1 = \{(A, B; y) : A = gh^y \wedge B = g^y\}.$$

For relation R_0 , apply AND-composition to:

$$R_{0A} = \{(A; y) : A = h^y\}, \quad R_{0B} = \{(B; z) : B = h^z\}.$$

For relation R_1 , apply EQ-composition to:

$$R_{1A} = \{(A; y) : A/g = h^y\}, \quad R_{1B} = \{(B; y) : B = g^y\}.$$

R_{0A} , R_{0B} , R_{1A} , R_{1B} are instances of (slight variations of) Schnorr's protocol.

Example 5.15

Σ -protocol needed for relation R :

$$R = \{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}.$$

Distinguish cases $x = 0$ and $x = 1$ for given $(A, B; x, y, z) \in R$.

If $x = 0$, then $A = h^y \wedge B = h^z$; if $x = 1$, then $A = gh^y \wedge B = g^y$.

For relation R , apply OR-composition to:

$$R_0 = \{(A, B; y, z) : A = h^y \wedge B = h^z\},$$

$$R_1 = \{(A, B; y) : A = gh^y \wedge B = g^y\}.$$

For relation R_0 , apply AND-composition to:

$$R_{0A} = \{(A; y) : A = h^y\}, \quad R_{0B} = \{(B; z) : B = h^z\}.$$

For relation R_1 , apply EQ-composition to:

$$R_{1A} = \{(A; y) : A/g = h^y\}, \quad R_{1B} = \{(B; y) : B = g^y\}.$$

$R_{0A}, R_{0B}, R_{1A}, R_{1B}$ are instances of (slight variations of) Schnorr's protocol.

Example 5.15

Σ -protocol needed for relation R :

$$R = \{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}.$$

Distinguish cases $x = 0$ and $x = 1$ for given $(A, B; x, y, z) \in R$.

If $x = 0$, then $A = h^y \wedge B = h^z$; if $x = 1$, then $A = gh^y \wedge B = g^y$.

For relation R , apply OR-composition to:

$$R_0 = \{(A, B; y, z) : A = h^y \wedge B = h^z\},$$

$$R_1 = \{(A, B; y) : A = gh^y \wedge B = g^y\}.$$

For relation R_0 , apply AND-composition to:

$$R_{0A} = \{(A; y) : A = h^y\}, \quad R_{0B} = \{(B; z) : B = h^z\}.$$

For relation R_1 , apply EQ-composition to:

$$R_{1A} = \{(A; y) : A/g = h^y\}, \quad R_{1B} = \{(B; y) : B = g^y\}.$$

$R_{0A}, R_{0B}, R_{1A}, R_{1B}$ are instances of (slight variations of) Schnorr's protocol.

Example 5.15

Σ -protocol needed for relation R :

$$R = \{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}.$$

Distinguish cases $x = 0$ and $x = 1$ for given $(A, B; x, y, z) \in R$.

If $x = 0$, then $A = h^y \wedge B = h^z$; if $x = 1$, then $A = gh^y \wedge B = g^y$.

For relation R , apply OR-composition to:

$$R_0 = \{(A, B; y, z) : A = h^y \wedge B = h^z\},$$

$$R_1 = \{(A, B; y) : A = gh^y \wedge B = g^y\}.$$

For relation R_0 , apply AND-composition to:

$$R_{0A} = \{(A; y) : A = h^y\}, \quad R_{0B} = \{(B; z) : B = h^z\}.$$

For relation R_1 , apply EQ-composition to:

$$R_{1A} = \{(A; y) : A/g = h^y\}, \quad R_{1B} = \{(B; y) : B = g^y\}.$$

R_{0A} , R_{0B} , R_{1A} , R_{1B} are instances of (slight variations of) Schnorr's protocol.

Figure 5.10 (Σ -protocol for $\{(A, B; x, y, z) : A = g^x h^y \wedge B = g^{xy} h^{(1-x)z} \wedge x \in \{0, 1\}\}$)

Prover	Verifier
$(\text{case } x = 0)$	$(\text{case } x = 1)$
$u_{0A}, u_{0B} \in_R \mathbb{Z}_n$	$u_1 \in_R \mathbb{Z}_n$
$a_{0A} \leftarrow h^{u_{0A}}$	$a_{1A} \leftarrow h^{u_1}$
$a_{0B} \leftarrow h^{u_{0B}}$	$a_{1B} \leftarrow g^{u_1}$
$c_1, r_1 \in_R \mathbb{Z}_n$	$c_0, r_{0A}, r_{0B} \in_R \mathbb{Z}_n$
$a_{1A} \leftarrow h^{r_1}(A/g)^{-c_1}$	$a_{0A} \leftarrow h^{r_{0A}} A^{-c_0}$
$a_{1B} \leftarrow g^{r_1} B^{-c_1}$	$a_{0B} \leftarrow h^{r_{0B}} B^{-c_0}$
$c_0 \leftarrow_n c - c_1$	$c_1 \leftarrow_n c - c_0$
$r_{0A} \leftarrow_n u_{0A} + c_0 y$	$r_1 \leftarrow_n u_1 + c_1 y$
$r_{0B} \leftarrow_n u_{0B} + c_0 z$	
	$\xrightarrow{a_{0A}, a_{0B}, a_{1A}, a_{1B}}$
	$\xleftarrow{c} c \in_R \mathbb{Z}_n$
	$\xrightarrow{c_0, c_1, r_{0A}, r_{0B}, r_1}$
	$c_0 + c_1 \stackrel{?}{=}_n c$
	$h^{r_{0A}} \stackrel{?}{=} a_{0A} A^{c_0}$
	$h^{r_{0B}} \stackrel{?}{=} a_{0B} B^{c_0}$
	$h^{r_1} \stackrel{?}{=} a_{1A}(A/g)^{c_1}$
	$g^{r_1} \stackrel{?}{=} a_{1B} B^{c_1}$



Exercise

Exercise 5.16

Prove that the protocol of Figure 5.10 is a Σ -protocol for relation R , as defined in Example 5.15.

Exercise 5.17 ($\log_g h$ unknown to anyone)

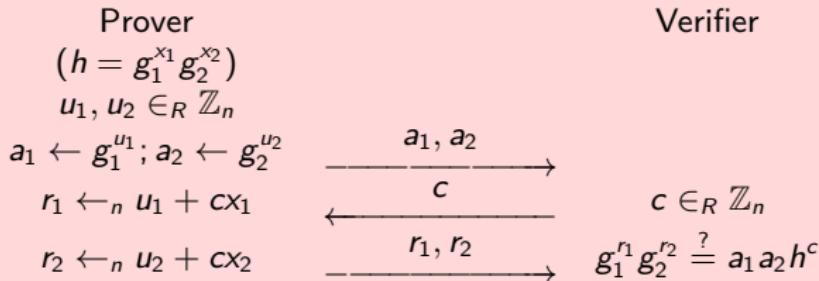
Design (and prove so!) Σ -protocol for $\{(B; x, y) : B = g^x h^y, \psi(x, y)\}$, where:

- (a) $\psi(x, y) \equiv \text{true}$;
- (b) $\psi(x, y) \equiv x = y$;
- (c) $\psi(x, y) \equiv \alpha x + \beta y = \gamma$ for given $\alpha, \beta \in \mathbb{Z}_n^*, \gamma \in \mathbb{Z}_n$;
- (d) $\psi(x, y) \equiv x \in \{0, 1\}$;
- (e) $\psi(x, y) \equiv x \in \{0, 1, \dots, 2^\ell - 1\}$, fixed integer ℓ , $1 \leq \ell \leq \lfloor \log_2 n \rfloor$;
- (f) $\psi(x, y) \equiv x \neq 0$;
- (g) $\psi(x, y) \equiv x \neq y$;
- (h) $\psi(x, y) \equiv \alpha x + \beta y \neq \gamma$ for given $\alpha, \beta \in \mathbb{Z}_n^*, \gamma \in \mathbb{Z}_n$;
- (i) $\psi(x, y) \equiv xy = 1$;
- (j) $\psi(x, y) \equiv \exists_{\chi \in \mathbb{Z}_n} x = \chi^2$;
- (k) $\psi(x, y) \equiv x^2 = y^2$.

Hints: (a) use Okamoto's protocol; (b) eliminate variable y using $x = y$; (c) eliminate variable y using given equation; (d) similar to OR-composition; (e) consider binary representation of x and use k instances of protocol of part (d); (f) use instance of Okamoto's protocol by isolating g in equation $B = g^x h^y$, cf. Eq. (5.1); (g) and (h) are generalizations of part (f); (i) isolate g in equation for B^y and use EQ-composition with equation for B ; (j) use AND-composition and EQ-composition; (k) eliminate one variable and use OR-composition.



Figure 5.11 (Alternative to Okamoto's protocol?)



Exercise 5.18

See Figure 4.5. Is Figure 5.11 a Σ -protocol for $\{(h; x_1, x_2) : h = g_1^{x_1} g_2^{x_2}\}$?

Exercise 5.19 ($\log_g h$ unknown to anyone)

Design Σ -protocols (and prove correctness) for relations:

- ③ $\{(A, B; x, y, z) : A = g^x h^y, B = g^{1/x} h^z, x \neq 0\}$;
- ④ $\{(A_1, A_2, B; x_1, x_2, y_1, y_2, z) : A_1 = g^{x_1} h^{y_1}, A_2 = g^{x_2} h^{y_2}, B = g^{x_1 x_2} h^z\}$.

Exercise 5.20 ($\log_g h$ unknown to anyone)

For Boolean variables v_1, \dots, v_ℓ , consider instance of 3SAT problem, given by Boolean formula Φ consisting of m clauses, which each consist of 3 literals:

$$\Phi = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge \cdots \wedge (l_{m,1} \vee l_{m,2} \vee l_{m,3}).$$

Each literal is of the form $l_{i,j} = v_k$ or $l_{i,j} = \bar{v}_k = 1 - v_k$ (negation of v_k), $1 \leq k \leq \ell$. Construct Σ -protocol for relation:

$$R'_{3SAT} = \{(\Phi, B_1, \dots, B_\ell; x_1, y_1, \dots, x_\ell, y_\ell) : \Phi(x_1, \dots, x_\ell), \\ \forall_{k=1}^\ell B_k = g^{x_k} h^{y_k}, x_k \in \{0, 1\}\}.$$

Exercise 5.21

See previous exercise. Σ -protocol for R'_{3SAT} actually proves knowledge of witnesses to open the commitments B_1, \dots, B_ℓ . Construct a more efficient way for proving that Φ is satisfiable, by considering instead relation:

$$R_{3SAT} = \{(\Phi; x_1, \dots, x_\ell) : \Phi(x_1, \dots, x_\ell)\}.$$

Schnorr Signatures

Cryptographic hash function H .

Fiat-Shamir heuristic

Transform identification scheme into digital signature scheme by “computing challenge as hash of **prover’s announcement** and **message to be signed**.”

For Schnorr’s protocol (Figure 4.3): put $c \leftarrow H(a, M)$.

Example (Schnorr signature scheme)

Key generation. Key pair $(h; x)$ with private key $x \in_R \mathbb{Z}_n$, public key $h \leftarrow g^x$.

Signature generation. On input of message M , private key x , set

$$u \in_R \mathbb{Z}_n; a \leftarrow g^u; c \leftarrow H(a, M); r \leftarrow_n u + cx.$$

Signature on M is pair (c, r) .

Signature verification. On input of message M , pair (c, r) , public key h , accept (c, r) as signature on M iff $c = H(g^r h^{-c}, M)$.

Schnorr signatures secure in random oracle model: challenge c is unpredictable.



Schnorr Signatures

Cryptographic hash function H .

Fiat-Shamir heuristic

Transform identification scheme into digital signature scheme by “computing challenge as hash of **prover’s announcement** and **message to be signed**.”

For Schnorr’s protocol (Figure 4.3): put $c \leftarrow H(a, M)$.

Example (Schnorr signature scheme)

Key generation. Key pair $(h; x)$ with private key $x \in_R \mathbb{Z}_n$, public key $h \leftarrow g^x$.

Signature generation. On input of message M , private key x , set

$$u \in_R \mathbb{Z}_n; a \leftarrow g^u; c \leftarrow H(a, M); r \leftarrow_n u + cx.$$

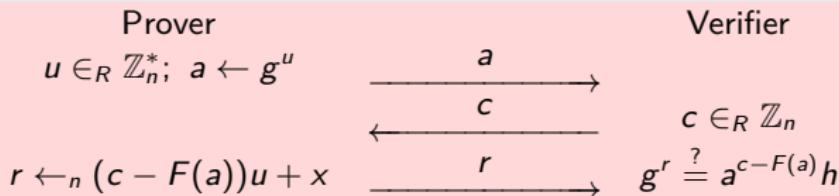
Signature on M is pair (c, r) .

Signature verification. On input of message M , pair (c, r) , public key h , accept (c, r) as signature on M iff $c = H(g^r h^{-c}, M)$.

Schnorr signatures secure in random oracle model: challenge c is unpredictable.



Figure 5.12 (Parametrized insecure variant of Schnorr's protocol)

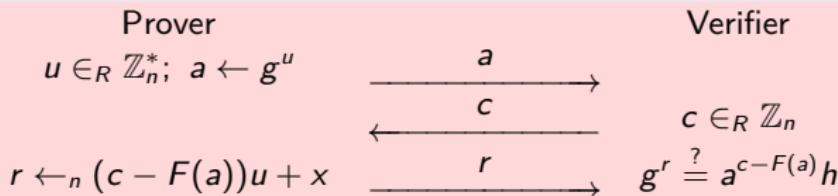


Exercise 5.22

To see that Fiat-Shamir heuristic does not necessarily lead to secure signature schemes, consider the protocol in Figure 5.12. Function $F : \langle g \rangle \rightarrow \mathbb{Z}_n$ can be your favorite hash function; for simplicity assumed that F maps into \mathbb{Z}_n . Note that function $F(w) = 0$ for $w \in \langle g \rangle$ yields the protocol of Exercise 5.4.

- ➊ Show that the protocol is complete, special sound, and honest-verifier zero-knowledge (for any function $F : \langle g \rangle \rightarrow \mathbb{Z}_n$).
- ➋ What happens if we generate the challenge as $c \leftarrow F(a)$ to obtain a non-interactive version of the protocol? That is, what happens if we instantiate the random oracle with $H = F$.

Figure 5.12 (Parametrized insecure variant of Schnorr's protocol)



Exercise 5.22

To see that Fiat-Shamir heuristic does not necessarily lead to secure signature schemes, consider the protocol in Figure 5.12. Function $F : \langle g \rangle \rightarrow \mathbb{Z}_n$ can be your favorite hash function; for simplicity assumed that F maps into \mathbb{Z}_n . Note that function $F(w) = 0$ for $w \in \langle g \rangle$ yields the protocol of Exercise 5.4.

- ❶ Show that the protocol is complete, special sound, and honest-verifier zero-knowledge (for any function $F : \langle g \rangle \rightarrow \mathbb{Z}_n$).
- ❷ What happens if we generate the challenge as $c \leftarrow F(a)$ to obtain a non-interactive version of the protocol? That is, what happens if we instantiate the random oracle with $H = F$.

Proofs of Validity

Fiat-Shamir heuristic can be applied to any Σ -protocol.
 Result: non-interactive Σ -proof.

Definition 5.23

Let H be a cryptographic hash function. For any Σ -protocol as in Figure 5.1, a (non-interactive) **Σ -proof for relation R** is defined in terms of two algorithms.

Proof generation. Given $(v; w) \in R$, a Σ -proof is a pair $(a; r)$, where $a = \alpha(v; w; u_P)$ and $r = \rho(v; w; H(a; v); u_P)$.

Proof verification. For $v \in V$, $(a; r)$ is accepted as Σ -proof if and only if $\varphi(v; a; H(a; v); r)$ holds.

Σ -proof consists of announcement a and response r .

Typically, size of Σ -proof reduced by replacing announcement a by challenge c .

Example 5.24 (Proof of validity)

Turn solution to Exercise 5.17(d) into Σ -proof.

Let $B = g^x h^y$ where $x \in \{0, 1\}$, $y \in \mathbb{Z}_n$ is prover's witness. Write $\bar{x} = 1 - x$.

Proof generation. $u_x, r_{\bar{x}}, c_{\bar{x}} \in_R \mathbb{Z}_n$; $a_x \leftarrow h^{u_x}$; $a_{\bar{x}} \leftarrow h^{r_{\bar{x}}} (B/g^{\bar{x}})^{-c_{\bar{x}}}$;
 $c \leftarrow H(a_0, a_1, B)$; $c_x \leftarrow_n c - c_{\bar{x}}$; $r_x \leftarrow_n u_x + c_x y$.
 Output (c_0, c_1, r_0, r_1) as proof.

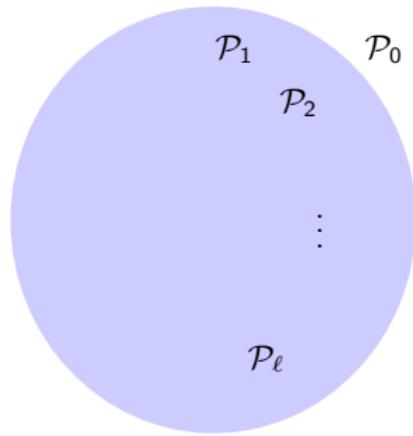
Proof verification. On input of proof (c_0, c_1, r_0, r_1) for commitment B , accept proof iff $c_0 + c_1 =_n H(h^{r_0} B^{-c_0}, h^{r_1} (B/g)^{-c_1}, B)$.

B needed in input to H to fix "context": otherwise proofs can be forged!

Exercise 5.25

Show how to forge a Σ -proof (c_0, c_1, r_0, r_1) for a commitment B if proof verification is changed into $c_0 + c_1 =_n H(h^{r_0} B^{-c_0}, h^{r_1} (B/g)^{-c_1})$ by making a suitable choice for B . Hint: B can be set after $c = H(a_0, a_1)$ is computed.

Anonymous Signatures: Group Signatures



Group manager: \mathcal{P}_0

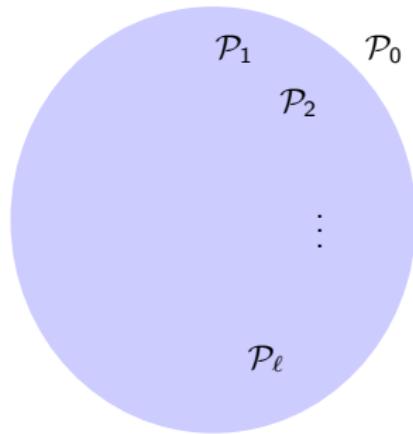
Group members: $\mathcal{P}_1, \dots, \mathcal{P}_l$

- Any group member \mathcal{P}_i can sign on behalf of the group.
- Actual signer remains **anonymous**.
- Anonymity is **revocable**: group manager can prove which group member produced a given signature.

Group public key: h

Party \mathcal{P}_i 's private key: x_i

Anonymous Signatures: Group Signatures



Group manager: \mathcal{P}_0

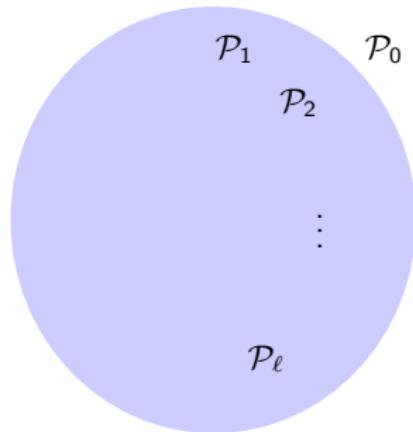
Group members: $\mathcal{P}_1, \dots, \mathcal{P}_\ell$

- Any group member \mathcal{P}_i can sign on behalf of the group.
- Actual signer remains **anonymous**.
- Anonymity is **revocable**: group manager can prove which group member produced a given signature.

Group public key: h

Party \mathcal{P}_i 's private key: x_i

Anonymous Signatures: Group Signatures



Group manager: \mathcal{P}_0

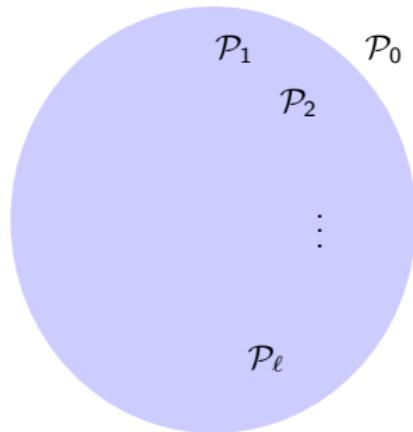
Group members: $\mathcal{P}_1, \dots, \mathcal{P}_\ell$

- Any group member \mathcal{P}_i can sign on behalf of the group.
- Actual signer remains **anonymous**.
- Anonymity is **revocable**: group manager can prove which group member produced a given signature.

Group public key: h

Party \mathcal{P}_i 's private key: x_i

Anonymous Signatures: Group Signatures



Group manager: \mathcal{P}_0

Group members: $\mathcal{P}_1, \dots, \mathcal{P}_\ell$

- Any group member \mathcal{P}_i can sign on behalf of the group.
- Actual signer remains **anonymous**.
- Anonymity is **revocable**: group manager can prove which group member produced a given signature.

Group public key: h

Party \mathcal{P}_i 's private key: x_i

Definition 5.26 (Group signature scheme)

Four components, involving group manager \mathcal{P}_0 , group members $\mathcal{P}_1, \dots, \mathcal{P}_\ell$.

Key generation. Protocol between $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_\ell$ for generating public key h for the group, private key x_0 for group manager \mathcal{P}_0 and private key x_i for each group member \mathcal{P}_i , $1 \leq i \leq \ell$.

Signature generation. Algorithm that on input of message M , public key h of the group, private key x_i of group member \mathcal{P}_i , outputs signature S .

Signature verification. Algorithm that on input of message M , public key h of the group, signature S , tests if S is valid group signature on M w.r.t. public key h .

Signature opening. Algorithm that on input of message M , public key h of the group, valid signature S , private key x_0 of group manager, outputs identity of group member who generated S .

Definition 5.26 (Group signature scheme)

Four components, involving group manager \mathcal{P}_0 , group members $\mathcal{P}_1, \dots, \mathcal{P}_\ell$.

Key generation. Protocol between $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_\ell$ for generating public key h for the group, private key x_0 for group manager \mathcal{P}_0 and private key x_i for each group member \mathcal{P}_i , $1 \leq i \leq \ell$.

Signature generation. Algorithm that on input of message M , public key h of the group, private key x_i of group member \mathcal{P}_i , outputs signature S .

Signature verification. Algorithm that on input of message M , public key h of the group, signature S , tests if S is valid group signature on M w.r.t. public key h .

Signature opening. Algorithm that on input of message M , public key h of the group, valid signature S , private key x_0 of group manager, outputs identity of group member who generated S .

Definition 5.26 (Group signature scheme)

Four components, involving group manager \mathcal{P}_0 , group members $\mathcal{P}_1, \dots, \mathcal{P}_\ell$.

Key generation. Protocol between $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_\ell$ for generating public key h for the group, private key x_0 for group manager \mathcal{P}_0 and private key x_i for each group member \mathcal{P}_i , $1 \leq i \leq \ell$.

Signature generation. Algorithm that on input of message M , public key h of the group, private key x_i of group member \mathcal{P}_i , outputs signature S .

Signature verification. Algorithm that on input of message M , public key h of the group, signature S , tests if S is valid group signature on M w.r.t. public key h .

Signature opening. Algorithm that on input of message M , public key h of the group, valid signature S , private key x_0 of group manager, outputs identity of group member who generated S .

Definition 5.26 (Group signature scheme)

Four components, involving group manager \mathcal{P}_0 , group members $\mathcal{P}_1, \dots, \mathcal{P}_\ell$.

Key generation. Protocol between $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_\ell$ for generating public key h for the group, private key x_0 for group manager \mathcal{P}_0 and private key x_i for each group member \mathcal{P}_i , $1 \leq i \leq \ell$.

Signature generation. Algorithm that on input of message M , public key h of the group, private key x_i of group member \mathcal{P}_i , outputs signature S .

Signature verification. Algorithm that on input of message M , public key h of the group, signature S , tests if S is valid group signature on M w.r.t. public key h .

Signature opening. Algorithm that on input of message M , public key h of the group, valid signature S , private key x_0 of group manager, outputs identity of group member who generated S .

Definition 5.26 (Group signature scheme)

Four components, involving group manager \mathcal{P}_0 , group members $\mathcal{P}_1, \dots, \mathcal{P}_\ell$.

Key generation. Protocol between $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_\ell$ for generating public key h for the group, private key x_0 for group manager \mathcal{P}_0 and private key x_i for each group member \mathcal{P}_i , $1 \leq i \leq \ell$.

Signature generation. Algorithm that on input of message M , public key h of the group, private key x_i of group member \mathcal{P}_i , outputs signature S .

Signature verification. Algorithm that on input of message M , public key h of the group, signature S , tests if S is valid group signature on M w.r.t. public key h .

Signature opening. Algorithm that on input of message M , public key h of the group, valid signature S , private key x_0 of group manager, outputs identity of group member who generated S .

Simple Construction

Party \mathcal{P}_i holds a discrete log key pair $(h_i; x_i)$ with $h_i = g^{x_i}$.

To sign a message, apply the Fiat-Shamir heuristic to Σ -protocol for proving knowledge of 1-out-of- ℓ private keys x_1, \dots, x_ℓ .

Exercise 5.27

Construct Σ -protocol for relation

$$R_{(1,\ell)} = \{(h_1, \dots, h_\ell; x) : \exists_{i=1}^{\ell} h_i = g^x\}$$

by generalizing OR-composition, and show it is indeed a Σ -protocol.

For revocability, include an encryption of the actual key pair used.



Simple Construction

Party \mathcal{P}_i holds a discrete log key pair $(h_i; x_i)$ with $h_i = g^{x_i}$.

To sign a message, apply the Fiat-Shamir heuristic to Σ -protocol for proving knowledge of 1-out-of- ℓ private keys x_1, \dots, x_ℓ .

Exercise 5.27

Construct Σ -protocol for relation

$$R_{(1,\ell)} = \{(h_1, \dots, h_\ell; x) : \exists_{i=1}^{\ell} h_i = g^x\}$$

by generalizing OR-composition, and show it is indeed a Σ -protocol.

For revocability, include an encryption of the actual key pair used.

Simple Construction

Party \mathcal{P}_i holds a discrete log key pair $(h_i; x_i)$ with $h_i = g^{x_i}$.

To sign a message, apply the Fiat-Shamir heuristic to Σ -protocol for proving knowledge of 1-out-of- ℓ private keys x_1, \dots, x_ℓ .

Exercise 5.27

Construct Σ -protocol for relation

$$R_{(1,\ell)} = \{(h_1, \dots, h_\ell; x) : \exists_{i=1}^{\ell} h_i = g^x\}$$

by generalizing OR-composition, and show it is indeed a Σ -protocol.

For revocability, include an encryption of the actual key pair used.

Example (ElGamal-based group signatures)

\mathcal{P}_i includes ElGamal encryption of h_i under \mathcal{P}_0 's public key in group signature.

Key generation. Each \mathcal{P}_i picks private key $x_i \in_R \mathbb{Z}_n$. Public key of group is $(h_0, h_1, \dots, h_\ell)$ with $h_i = g^{x_i}$.

Signature generation. Group member \mathcal{P}_i computes $(A, B) = (g^u, h_0^u h_i)$ with $u \in_R \mathbb{Z}_n$, and Σ -proof for $R'_{(1,i)}$ showing (A, B) encrypts one of h_1, \dots, h_ℓ for which \mathcal{P}_i knows private key:

$$R'_{(1,i)} = \{(A, B, h_0, h_1, \dots, h_\ell; u, x) : A = g^u, B = h_0^u g^{x_i}, \exists_{j=1}^\ell h_j = g^{x_j}\}.$$

Signature verification. Σ -proof contained in group signature is verified.

Signature opening. Group manager \mathcal{P}_0 decrypts ElGamal encryption (A, B) contained in group signature and proves correctness: \mathcal{P}_0 outputs $d = A^{x_0}$ and Σ -proof showing $\log_{B_A} h_0 = \log_{B_A} d$. Anyone may now compute B/d , which will match the public key of the group member who produced the signature.

Example (ElGamal-based group signatures)

\mathcal{P}_i includes ElGamal encryption of h_i under \mathcal{P}_0 's public key in group signature.

Key generation. Each \mathcal{P}_i picks private key $x_i \in_R \mathbb{Z}_n$. Public key of group is $(h_0, h_1, \dots, h_\ell)$ with $h_i = g^{x_i}$.

Signature generation. Group member \mathcal{P}_i computes $(A, B) = (g^u, h_0^u h_i)$ with $u \in_R \mathbb{Z}_n$, and Σ -proof for $R'_{(1,\ell)}$ showing (A, B) encrypts one of h_1, \dots, h_ℓ for which \mathcal{P}_i knows private key:

$$R'_{(1,\ell)} = \{(A, B, h_0, h_1, \dots, h_\ell; u, x) : A = g^u, B = h_0^u g^x, \exists_{i=1}^\ell h_i = g^{x_i}\}.$$

Signature verification. Σ -proof contained in group signature is verified.

Signature opening. Group manager \mathcal{P}_0 decrypts ElGamal encryption (A, B) contained in group signature and proves correctness: \mathcal{P}_0 outputs $d = A^{x_0}$ and Σ -proof showing $\log_g h_0 = \log_A d$. Anyone may now compute B/d , which will match the public key of the group member who produced the signature.

Example (ElGamal-based group signatures)

\mathcal{P}_i includes ElGamal encryption of h_i under \mathcal{P}_0 's public key in group signature.

Key generation. Each \mathcal{P}_i picks private key $x_i \in_R \mathbb{Z}_n$. Public key of group is $(h_0, h_1, \dots, h_\ell)$ with $h_i = g^{x_i}$.

Signature generation. Group member \mathcal{P}_i computes $(A, B) = (g^u, h_0^u h_i)$ with $u \in_R \mathbb{Z}_n$, and Σ -proof for $R'_{(1,\ell)}$ showing (A, B) encrypts one of h_1, \dots, h_ℓ for which \mathcal{P}_i knows private key:

$$R'_{(1,\ell)} = \{(A, B, h_0, h_1, \dots, h_\ell; u, x) : A = g^u, B = h_0^u g^x, \exists_{i=1}^\ell h_i = g^{x_i}\}.$$

Signature verification. Σ -proof contained in group signature is verified.

Signature opening. Group manager \mathcal{P}_0 decrypts ElGamal encryption (A, B) contained in group signature and proves correctness: \mathcal{P}_0 outputs $d = A^{x_0}$ and Σ -proof showing $\log_g h_0 = \log_A d$. Anyone may now compute B/d , which will match the public key of the group member who produced the signature.

Example (ElGamal-based group signatures)

\mathcal{P}_i includes ElGamal encryption of h_i under \mathcal{P}_0 's public key in group signature.

Key generation. Each \mathcal{P}_i picks private key $x_i \in_R \mathbb{Z}_n$. Public key of group is $(h_0, h_1, \dots, h_\ell)$ with $h_i = g^{x_i}$.

Signature generation. Group member \mathcal{P}_i computes $(A, B) = (g^u, h_0^u h_i)$ with $u \in_R \mathbb{Z}_n$, and Σ -proof for $R'_{(1,\ell)}$ showing (A, B) encrypts one of h_1, \dots, h_ℓ for which \mathcal{P}_i knows private key:

$$R'_{(1,\ell)} = \{(A, B, h_0, h_1, \dots, h_\ell; u, x) : A = g^u, B = h_0^u g^x, \exists_{i=1}^\ell h_i = g^{x_i}\}.$$

Signature verification. Σ -proof contained in group signature is verified.

Signature opening. Group manager \mathcal{P}_0 decrypts ElGamal encryption (A, B) contained in group signature and proves correctness: \mathcal{P}_0 outputs $d = A^{x_0}$ and Σ -proof showing $\log_g h_0 = \log_A d$. Anyone may now compute B/d , which will match the public key of the group member who produced the signature.

Example (ElGamal-based group signatures)

\mathcal{P}_i includes ElGamal encryption of h_i under \mathcal{P}_0 's public key in group signature.

Key generation. Each \mathcal{P}_i picks private key $x_i \in_R \mathbb{Z}_n$. Public key of group is $(h_0, h_1, \dots, h_\ell)$ with $h_i = g^{x_i}$.

Signature generation. Group member \mathcal{P}_i computes $(A, B) = (g^u, h_0^u h_i)$ with $u \in_R \mathbb{Z}_n$, and Σ -proof for $R'_{(1,\ell)}$ showing (A, B) encrypts one of h_1, \dots, h_ℓ for which \mathcal{P}_i knows private key:

$$R'_{(1,\ell)} = \{(A, B, h_0, h_1, \dots, h_\ell; u, x) : A = g^u, B = h_0^u g^x, \exists_{i=1}^\ell h_i = g^{x_i}\}.$$

Signature verification. Σ -proof contained in group signature is verified.

Signature opening. Group manager \mathcal{P}_0 decrypts ElGamal encryption (A, B) contained in group signature and proves correctness: \mathcal{P}_0 outputs $d = A^{x_0}$ and Σ -proof showing $\log_g h_0 = \log_A d$. Anyone may now compute B/d , which will match the public key of the group member who produced the signature.

Exercise

Exercise 5.28

Give a Σ -protocol for relation $R'_{(1,\ell)}$ and prove its correctness, in each of the following cases: (i) $\ell = 1$, (ii) $\ell = 2$, and (iii) arbitrary $\ell \geq 1$.

5-Way Latch



(1,5)-threshold scheme

5 padlocks “in series”:

**opening any 1 padlock
will open gate.**

Threshold cryptography (group-oriented cryptography)

Techniques to **distribute** cryptographic schemes **between multiple parties**.

Examples:

- Safe-deposit box requires use of **two keys**, one kept by the owner of the box and one kept by the bank.
- Control of nuclear weapons in Russia involves a "**2-out-of-3**" **access mechanism**, where the three parties are the President, the Defense Minister and the Defense Ministry (Time Magazine, May 4, 1992, p.13).
- Distributing power to issue digital signatures (e.g., issuance of root certificates in public key infrastructures).
 - Group signature \approx 1-out-of- ℓ threshold signature.
- Distributing power to decrypt messages (key escrow, key recovery).
- Electronic voting: distributing power to decrypt votes.

6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Example 6.1 (See also this [Mathematica example](#).)

RSA cryptosystem, public exponent $e = 3$, modulus $m = pq$, $\gcd(e, \phi(m)) = 1$. Two persons split private key $d = 1/e \bmod \phi(m)$ into **most-significant half** and **least-significant half**. How secure?

Since $0 < d < \phi(m)$, $3d = 1 + l\phi(m)$ holds either for $l = 1$ or $l = 2$. Since p and q not divisible by 3, $\phi(m) \not\equiv 2 \pmod{3}$, hence $l \equiv 2 \pmod{3}$. Thus $l = 2$.

$$\text{Approximation } \hat{d} = \left\lfloor \frac{1+2(m-2\sqrt{m}+1)}{3} \right\rfloor \text{ for } d = \frac{1+2\phi(m)}{3}.$$

Approximation error: $\hat{d} - d = \left\lfloor \frac{2}{3}(p+q-2\sqrt{m}) \right\rfloor$. Since $p+q > 2\sqrt{m}$ (use $(\sqrt{p}-\sqrt{q})^2 > 0$):

$$0 \leq \hat{d} - d < \sqrt{m},$$

as p and q of equal bit length.

Thus, most-significant half of d matches most-significant half of \hat{d} .

Person receiving least-significant half of d is able to construct all of d 's bits!

6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Example 6.1 (See also this [Mathematica example](#).)

RSA cryptosystem, public exponent $e = 3$, modulus $m = pq$, $\gcd(e, \phi(m)) = 1$. Two persons split private key $d = 1/e \bmod \phi(m)$ into **most-significant half** and **least-significant half**. How secure?

Since $0 < d < \phi(m)$, $3d = 1 + l\phi(m)$ holds either for $l = 1$ or $l = 2$. Since p and q not divisible by 3, $\phi(m) \not\equiv 2 \pmod{3}$, hence $l \equiv 2 \pmod{3}$. Thus $l = 2$.

$$\text{Approximation } \hat{d} = \left\lfloor \frac{1+2(m-2\sqrt{m}+1)}{3} \right\rfloor \quad \text{for } d = \frac{1+2\phi(m)}{3}.$$

Approximation error: $\hat{d} - d = \left\lfloor \frac{2}{3}(p+q-2\sqrt{m}) \right\rfloor$. Since $p+q > 2\sqrt{m}$ (use $(\sqrt{p}-\sqrt{q})^2 > 0$):

$$0 \leq \hat{d} - d < \sqrt{m},$$

as p and q of equal bit length.

Thus, most-significant half of d matches most-significant half of \hat{d} .

Person receiving least-significant half of d is able to construct all of d 's bits!



6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Example 6.1 (See also this [Mathematica example](#).)

RSA cryptosystem, public exponent $e = 3$, modulus $m = pq$, $\gcd(e, \phi(m)) = 1$. Two persons split private key $d = 1/e \bmod \phi(m)$ into **most-significant half** and **least-significant half**. How secure?

Since $0 < d < \phi(m)$, $3d = 1 + l\phi(m)$ holds either for $l = 1$ or $l = 2$. Since p and q not divisible by 3, $\phi(m) \not\equiv 2 \pmod{3}$, hence $l \equiv 2 \pmod{3}$. Thus $l = 2$.

$$\text{Approximation} \quad \hat{d} = \left\lfloor \frac{1+2(m-2\sqrt{m}+1)}{3} \right\rfloor \quad \text{for } d = \frac{1+2\phi(m)}{3}.$$

Approximation error: $\hat{d} - d = \left\lfloor \frac{2}{3}(p+q-2\sqrt{m}) \right\rfloor$. Since $p+q > 2\sqrt{m}$ (use $(\sqrt{p}-\sqrt{q})^2 > 0$):

$$0 \leq \hat{d} - d < \sqrt{m},$$

as p and q of equal bit length.

Thus, most-significant half of d matches most-significant half of \hat{d} .

Person receiving least-significant half of d is able to construct all of d 's bits!



6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Example 6.1 (See also this [Mathematica example](#).)

RSA cryptosystem, public exponent $e = 3$, modulus $m = pq$, $\gcd(e, \phi(m)) = 1$. Two persons split private key $d = 1/e \bmod \phi(m)$ into **most-significant half** and **least-significant half**. How secure?

Since $0 < d < \phi(m)$, $3d = 1 + l\phi(m)$ holds either for $l = 1$ or $l = 2$. Since p and q not divisible by 3, $\phi(m) \not\equiv 2 \pmod{3}$, hence $l \equiv 2 \pmod{3}$. Thus $l = 2$.

Approximation $\hat{d} = \left\lfloor \frac{1+2(m-2\sqrt{m}+1)}{3} \right\rfloor$ for $d = \frac{1+2\phi(m)}{3}$.

Approximation error: $\hat{d} - d = \left\lfloor \frac{2}{3}(p + q - 2\sqrt{m}) \right\rfloor$. Since $p + q > 2\sqrt{m}$ (use $(\sqrt{p} - \sqrt{q})^2 > 0$):

$$0 \leq \hat{d} - d < \sqrt{m},$$

as p and q of equal bit length.

Thus, most-significant half of d matches most-significant half of \hat{d} .

Person receiving least-significant half of d is able to construct all of d 's bits!

6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Example 6.1 (See also this [Mathematica example](#).)

RSA cryptosystem, public exponent $e = 3$, modulus $m = pq$, $\gcd(e, \phi(m)) = 1$. Two persons split private key $d = 1/e \bmod \phi(m)$ into **most-significant half** and **least-significant half**. How secure?

Since $0 < d < \phi(m)$, $3d = 1 + l\phi(m)$ holds either for $l = 1$ or $l = 2$. Since p and q not divisible by 3, $\phi(m) \not\equiv 2 \pmod{3}$, hence $l \equiv 2 \pmod{3}$. Thus $l = 2$.

Approximation $\hat{d} = \left\lfloor \frac{1+2(m-2\sqrt{m}+1)}{3} \right\rfloor$ for $d = \frac{1+2\phi(m)}{3}$.

Approximation error: $\hat{d} - d = \left\lfloor \frac{2}{3}(p + q - 2\sqrt{m}) \right\rfloor$. Since $p + q > 2\sqrt{m}$ (use $(\sqrt{p} - \sqrt{q})^2 > 0$):

$$0 \leq \hat{d} - d < \sqrt{m},$$

as p and q of equal bit length.

Thus, most-significant half of d matches most-significant half of \hat{d} .

Person receiving least-significant half of d is able to construct all of d 's bits!

6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Example 6.1 (See also this [Mathematica example](#).)

RSA cryptosystem, public exponent $e = 3$, modulus $m = pq$, $\gcd(e, \phi(m)) = 1$. Two persons split private key $d = 1/e \bmod \phi(m)$ into **most-significant half** and **least-significant half**. How secure?

Since $0 < d < \phi(m)$, $3d = 1 + l\phi(m)$ holds either for $l = 1$ or $l = 2$. Since p and q not divisible by 3, $\phi(m) \not\equiv 2 \pmod{3}$, hence $l \equiv 2 \pmod{3}$. Thus $l = 2$.

Approximation $\hat{d} = \left\lfloor \frac{1+2(m-2\sqrt{m}+1)}{3} \right\rfloor$ for $d = \frac{1+2\phi(m)}{3}$.

Approximation error: $\hat{d} - d = \left\lfloor \frac{2}{3}(p + q - 2\sqrt{m}) \right\rfloor$. Since $p + q > 2\sqrt{m}$ (use $(\sqrt{p} - \sqrt{q})^2 > 0$):

$$0 \leq \hat{d} - d < \sqrt{m},$$

as p and q of equal bit length.

Thus, most-significant half of d matches most-significant half of \hat{d} .

Person receiving least-significant half of d is able to construct all of d 's bits!

6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Splitting the Bit

Breaking bit string into pieces not a secure way to share a secret.
And, how do we “split the bit”?

Critical step: use additional randomness.

Splitting secret bit $s \in \{0, 1\}$ into shares $s_1, s_2 \in \{0, 1\}$

- To split s into s_1, s_2 : pick $u \in_R \{0, 1\}$, set $s_1 = s \oplus u$ and $s_2 = u$.
- To recover s from s_1, s_2 : compute $s_1 \oplus s_2 = s \oplus u \oplus u = s$.

Neither s_1 nor s_2 on its own reveals any information on s .

Exercise 6.2

Suppose u is uniformly distributed. Show that s_1 and s_2 are also uniformly distributed, irrespective of the distribution of s .

6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Splitting the Bit

Breaking bit string into pieces not a secure way to share a secret.
And, how do we “split the bit”?

Critical step: use additional randomness.

Splitting secret bit $s \in \{0, 1\}$ into shares $s_1, s_2 \in \{0, 1\}$

- To split s into s_1, s_2 : pick $u \in_R \{0, 1\}$, set $s_1 = s \oplus u$ and $s_2 = u$.
- To recover s from s_1, s_2 : compute $s_1 \oplus s_2 = s \oplus u \oplus u = s$.

Neither s_1 nor s_2 on its own reveals any information on s .

Exercise 6.2

Suppose u is uniformly distributed. Show that s_1 and s_2 are also uniformly distributed, irrespective of the distribution of s .

Splitting the Bit

Breaking bit string into pieces not a secure way to share a secret.
And, how do we “split the bit”?

Critical step: use additional randomness.

Splitting secret bit $s \in \{0, 1\}$ into shares $s_1, s_2 \in \{0, 1\}$

- To split s into s_1, s_2 : pick $u \in_R \{0, 1\}$, set $s_1 = s \oplus u$ and $s_2 = u$.
- To recover s from s_1, s_2 : compute $s_1 \oplus s_2 = s \oplus u \oplus u = s$.

Neither s_1 nor s_2 on its own reveals any information on s .

Exercise 6.2

Suppose u is uniformly distributed. Show that s_1 and s_2 are also uniformly distributed, irrespective of the distribution of s .

Splitting the Bit

Breaking bit string into pieces not a secure way to share a secret.
And, how do we “split the bit”?

Critical step: use additional randomness.

Splitting secret bit $s \in \{0, 1\}$ into shares $s_1, s_2 \in \{0, 1\}$

- To split s into s_1, s_2 : pick $u \in_R \{0, 1\}$, set $s_1 = s \oplus u$ and $s_2 = u$.
- To recover s from s_1, s_2 : compute $s_1 \oplus s_2 = s \oplus u \oplus u = s$.

Neither s_1 nor s_2 on its own reveals any information on s .

Exercise 6.2

Suppose u is uniformly distributed. Show that s_1 and s_2 are also uniformly distributed, irrespective of the distribution of s .

Splitting the Bit

Breaking bit string into pieces not a secure way to share a secret.
And, how do we “split the bit”?

Critical step: use additional randomness.

Splitting secret bit $s \in \{0, 1\}$ into shares $s_1, s_2 \in \{0, 1\}$

- To split s into s_1, s_2 : pick $u \in_R \{0, 1\}$, set $s_1 = s \oplus u$ and $s_2 = u$.
- To recover s from s_1, s_2 : compute $s_1 \oplus s_2 = s \oplus u \oplus u = s$.

Neither s_1 nor s_2 on its own reveals any information on s .

Exercise 6.2

Suppose u is uniformly distributed. Show that s_1 and s_2 are also uniformly distributed, irrespective of the distribution of s .

Splitting the Bit

Breaking bit string into pieces not a secure way to share a secret.
And, how do we “split the bit”?

Critical step: use additional randomness.

Splitting secret bit $s \in \{0, 1\}$ into shares $s_1, s_2 \in \{0, 1\}$

- To split s into s_1, s_2 : pick $u \in_R \{0, 1\}$, set $s_1 = s \oplus u$ and $s_2 = u$.
- To recover s from s_1, s_2 : compute $s_1 \oplus s_2 = s \oplus u \oplus u = s$.

Neither s_1 nor s_2 on its own reveals any information on s .

Exercise 6.2

Suppose u is uniformly distributed. Show that s_1 and s_2 are also uniformly distributed, irrespective of the distribution of s .

Splitting the Bit

Breaking bit string into pieces not a secure way to share a secret.
And, how do we “split the bit”?

Critical step: use additional randomness.

Splitting secret bit $s \in \{0, 1\}$ into shares $s_1, s_2 \in \{0, 1\}$

- To split s into s_1, s_2 : pick $u \in_R \{0, 1\}$, set $s_1 = s \oplus u$ and $s_2 = u$.
- To recover s from s_1, s_2 : compute $s_1 \oplus s_2 = s \oplus u \oplus u = s$.

Neither s_1 nor s_2 on its own reveals any information on s .

Exercise 6.2

Suppose u is uniformly distributed. Show that s_1 and s_2 are also uniformly distributed, irrespective of the distribution of s .

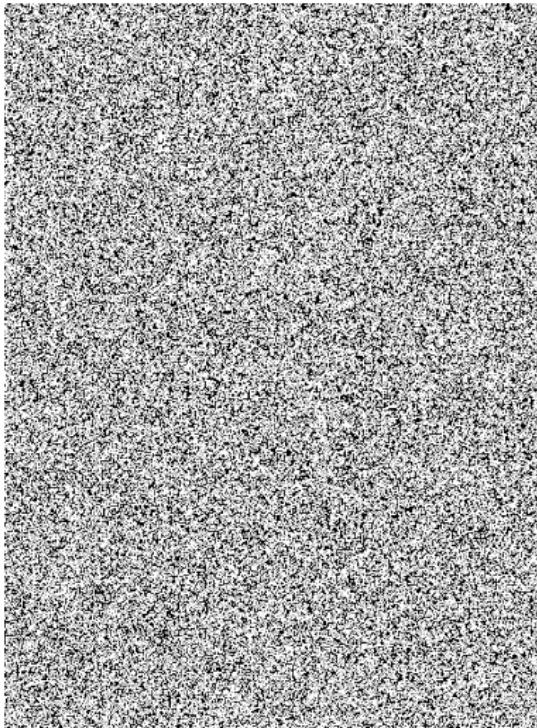
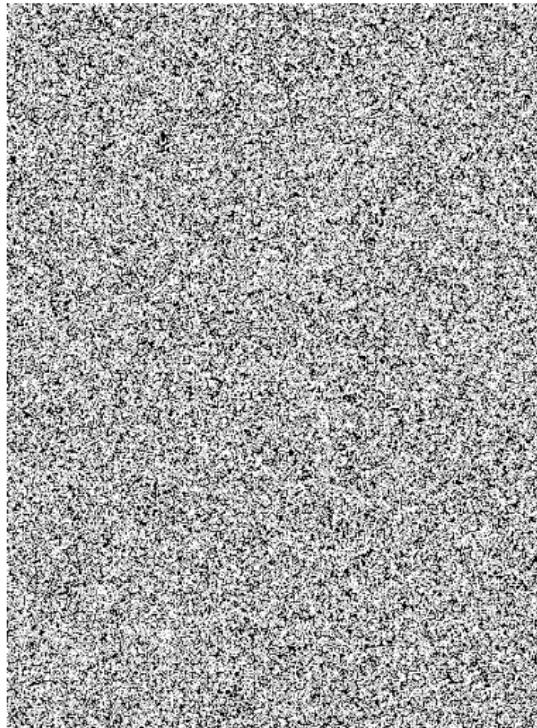
6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Visual Secret Sharing (Naor & Shamir, Eurocrypt 1994)



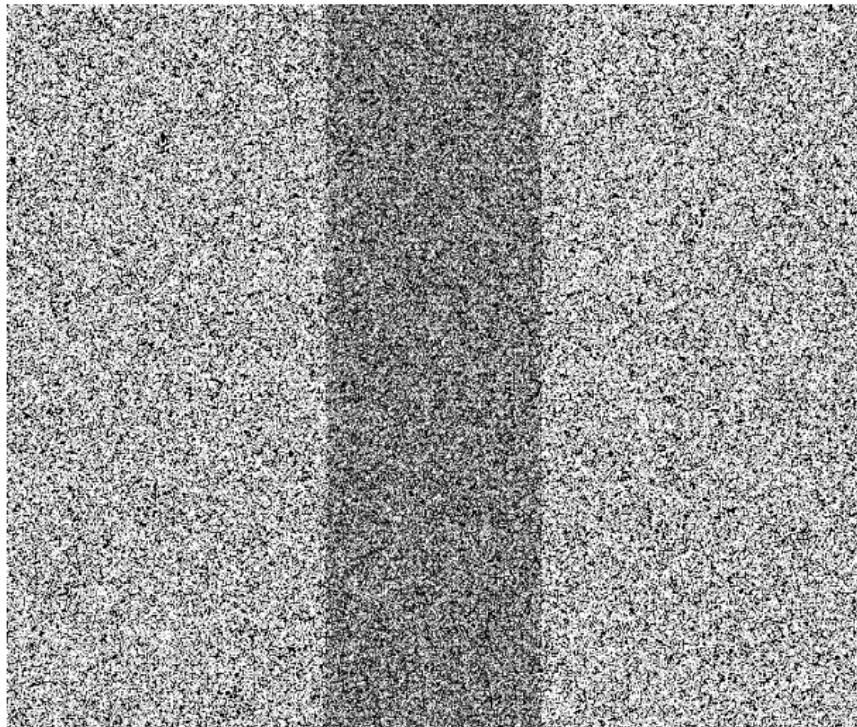
6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Visual Secret Sharing (Naor & Shamir, Eurocrypt 1994)



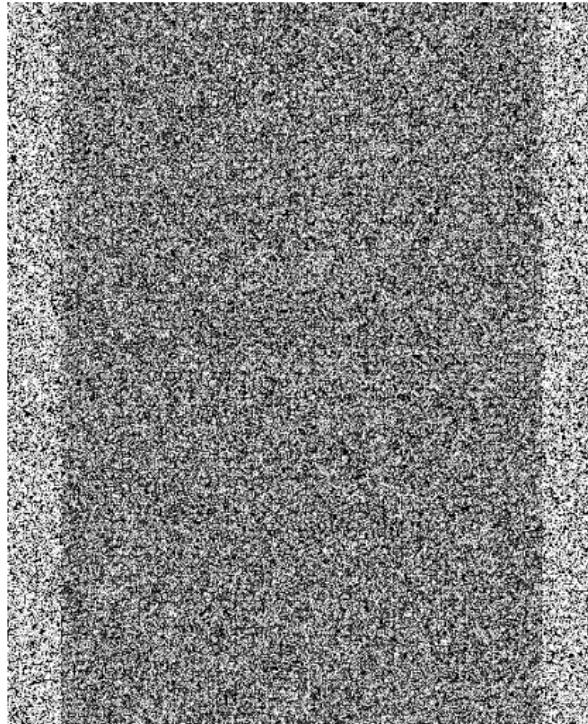
6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Visual Secret Sharing (Naor & Shamir, Eurocrypt 1994)



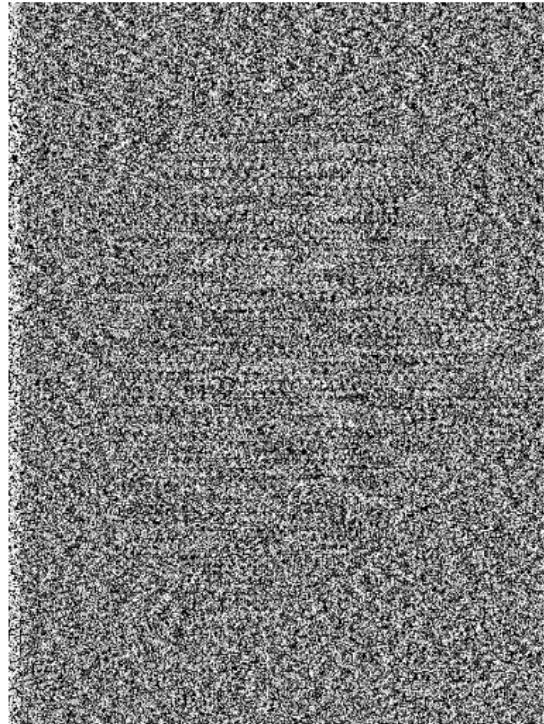
6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Visual Secret Sharing (Naor & Shamir, Eurocrypt 1994)



6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Visual Secret Sharing (Naor & Shamir, Eurocrypt 1994)



6.1 Secret Sharing

6.2 Verifiable Secret Sharing

6.3 Threshold Cryptosystems

6.1.1 Shamir's Threshold Scheme

Definition 6.3 (Secret sharing scheme)

Two protocols, involving **dealer \mathcal{D}** and **participants $\mathcal{P}_1, \dots, \mathcal{P}_\ell$** .

Distribution. Protocol in which dealer \mathcal{D} shares secret s such that each participant \mathcal{P}_i obtains share s_i , $1 \leq i \leq \ell$.

Reconstruction. Protocol in which secret s is recovered by pooling shares s_i , $i \in Q$, of any **qualified** set of participants $Q \subseteq \{\mathcal{P}_1, \dots, \mathcal{P}_\ell\}$.

Security requirements for secret sharing scheme:

- (i) any qualified set of participants can determine s by pooling their shares,
- (ii) any non-qualified set of participants cannot find *any* information on s when pooling their shares.



Definition 6.3 (Secret sharing scheme)

Two protocols, involving **dealer \mathcal{D}** and **participants $\mathcal{P}_1, \dots, \mathcal{P}_\ell$** .

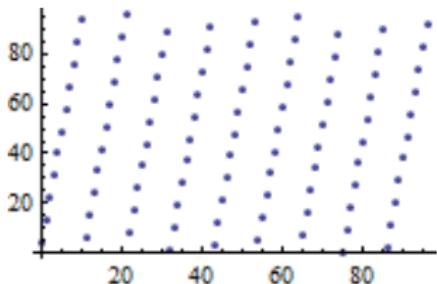
Distribution. Protocol in which dealer \mathcal{D} shares secret s such that each participant \mathcal{P}_i obtains share s_i , $1 \leq i \leq \ell$.

Reconstruction. Protocol in which secret s is recovered by pooling shares s_i , $i \in Q$, of any **qualified** set of participants $Q \subseteq \{\mathcal{P}_1, \dots, \mathcal{P}_\ell\}$.

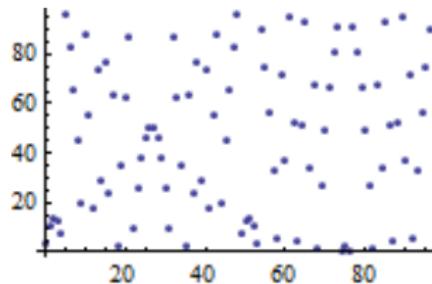
Security requirements for secret sharing scheme:

- (i) any qualified set of participants can determine s by pooling their shares,
- (ii) any non-qualified set of participants cannot find *any* information on s when pooling their shares.

Polynomials over Finite Field \mathbb{Z}_p



straight line ($\text{mod } 97$)

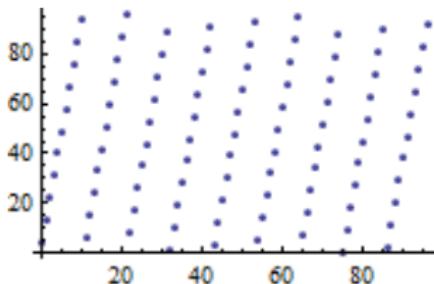


parabola ($\text{mod } 97$)

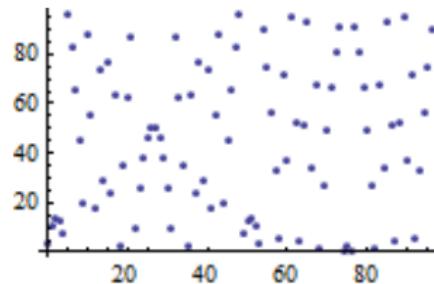
Mathematically, same behavior as polynomials over \mathbb{R} :

- Polynomial of degree $d > 0$ has at most d roots.
- Polynomial of degree d is uniquely determined by $d + 1$ points.

Polynomials over Finite Field \mathbb{Z}_p



straight line ($\text{mod } 97$)



parabola ($\text{mod } 97$)

Mathematically, same behavior as polynomials over \mathbb{R} :

- Polynomial of degree $d > 0$ has at most d roots.
- Polynomial of degree d is uniquely determined by $d + 1$ points.

Threshold Secret Sharing Scheme

Participants $\mathcal{P}_1, \dots, \mathcal{P}_\ell$.

Threshold t , $1 \leq t \leq \ell$.

(t, ℓ) -threshold access structure $\Gamma = \{Q \subseteq \{\mathcal{P}_1, \dots, \mathcal{P}_\ell\} : |Q| \geq t\}$.

Perfect (t, ℓ) threshold secret sharing scheme:

- Any group of t participants is able to recover the secret.
- No group of $t - 1$ or less participants can do so:
 - $t - 1$ or less participants **no information** on the secret.

Shamir's (t, ℓ) -threshold scheme

For secret $s \in \mathbb{Z}_p$.

Distribution. Dealer picks random polynomial $a(X) \in_R \mathbb{Z}_p[X]$ of degree $< t$ satisfying $a(0) = s$, and sends share $s_i = a(i)$ to participant \mathcal{P}_i , for $i = 1, \dots, \ell$.

Reconstruction. Any set Q of t participants may recover secret s from their shares by Lagrange interpolation:^a

$$s = \sum_{i \in Q} s_i \lambda_{Q,i}, \quad \text{with } \lambda_{Q,i} = \prod_{j \in Q \setminus \{i\}} \frac{j}{j - i}.$$

^aNote: $i \in Q$ shorthand for $\mathcal{P}_i \in Q$.

Example ((2, 5)-threshold scheme)

Distribution. Let $a(X) = s + uX$ with $u \in_R \mathbb{Z}_p$. Send share $s_i = a(i)$ to \mathcal{P}_i .

Reconstruction. Any pair $\mathcal{P}_i, \mathcal{P}_j$ recovers secret as $s = (is_j - js_i)/(i - j)$.

Shamir's (t, ℓ) -threshold scheme

For secret $s \in \mathbb{Z}_p$.

Distribution. Dealer picks random polynomial $a(X) \in_R \mathbb{Z}_p[X]$ of degree $< t$ satisfying $a(0) = s$, and sends share $s_i = a(i)$ to participant \mathcal{P}_i , for $i = 1, \dots, \ell$.

Reconstruction. Any set Q of t participants may recover secret s from their shares by Lagrange interpolation:^a

$$s = \sum_{i \in Q} s_i \lambda_{Q,i}, \quad \text{with } \lambda_{Q,i} = \prod_{j \in Q \setminus \{i\}} \frac{j}{j - i}.$$

^aNote: $i \in Q$ shorthand for $\mathcal{P}_i \in Q$.

Example ((2, 5)-threshold scheme)

Distribution. Let $a(X) = s + uX$ with $u \in_R \mathbb{Z}_p$. Send share $s_i = a(i)$ to \mathcal{P}_i .

Reconstruction. Any pair $\mathcal{P}_i, \mathcal{P}_j$ recovers secret as $s = (is_j - js_i)/(i - j)$.

Secret sharing schemes resists **passive** attacks only.

Verifiable secret sharing (VSS)

VSS scheme resists (combinations of) two types of **active** attacks:

- Dealer sending incorrect shares to some/all participants during distribution.
- Participants submitting incorrect shares during reconstruction.

Shamir's scheme is not a VSS scheme:

- During distribution, no guarantee shares s_i correspond to single polynomial $a(X)$ of degree $< t$.
- During reconstruction, no guarantee share s_i provided by \mathcal{P}_i is correct.
 - Nothing prevents \mathcal{P}_i from using $\tilde{s}_i \in_R \mathbb{Z}_p$ instead; reconstructed value \tilde{s} will be useless, and if only \mathcal{P}_i is cheating, \mathcal{P}_i finds value of s using the other $t - 1$ correct shares.

Secret sharing schemes resists **passive** attacks only.

Verifiable secret sharing (VSS)

VSS scheme resists (combinations of) two types of **active** attacks:

- Dealer sending incorrect shares to some/all participants during distribution.
- Participants submitting incorrect shares during reconstruction.

Shamir's scheme is not a VSS scheme:

- During distribution, no guarantee shares s_i correspond to single polynomial $a(X)$ of degree $< t$.
- During reconstruction, no guarantee share s_i provided by \mathcal{P}_i is correct.
 - Nothing prevents \mathcal{P}_i from using $\tilde{s}_i \in_R \mathbb{Z}_p$ instead; reconstructed value \tilde{s} will be useless, and if only \mathcal{P}_i is cheating, \mathcal{P}_i finds value of s using the other $t - 1$ correct shares.

Discrete log setting $\langle g \rangle$.

Feldman's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomial ($u_0 = s$):

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n.$$

Dealer sends shares $s_i = a(i)$ to \mathcal{P}_i in private, for $i = 1, \dots, \ell$, and broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i , \mathcal{P}_i checks:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \quad (6.1)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.1). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Commitments B_j define **unique** polynomial with coefficients $\log_g B_j$.

Discrete log setting $\langle g \rangle$.

Feldman's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomial ($u_0 = s$):

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n.$$

Dealer sends shares $s_i = a(i)$ to \mathcal{P}_i in private, for $i = 1, \dots, \ell$, and broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i , \mathcal{P}_i checks:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \quad (6.1)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.1). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Commitments B_j define **unique** polynomial with coefficients $\log_g B_j$.

Discrete log setting $\langle g \rangle$.

Feldman's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomial ($u_0 = s$):

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n.$$

Dealer sends shares $s_i = a(i)$ to \mathcal{P}_i in private, for $i = 1, \dots, \ell$, and broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i , \mathcal{P}_i checks:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \quad (6.1)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.1). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Commitments B_j define **unique** polynomial with coefficients $\log_g B_j$.

Discrete log setting $\langle g \rangle$.

Feldman's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomial ($u_0 = s$):

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n.$$

Dealer sends shares $s_i = a(i)$ to \mathcal{P}_i in private, for $i = 1, \dots, \ell$, and broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i , \mathcal{P}_i checks:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \quad (6.1)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.1). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Commitments B_j define **unique** polynomial with coefficients $\log_g B_j$.

Discrete log setting $\langle g \rangle$.

Feldman's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomial ($u_0 = s$):

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n.$$

Dealer sends shares $s_i = a(i)$ to \mathcal{P}_i in private, for $i = 1, \dots, \ell$, and broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i , \mathcal{P}_i checks:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \quad (6.1)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.1). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Commitments B_j define **unique** polynomial with coefficients $\log_g B_j$.

Discrete log setting $\langle g \rangle$.

Feldman's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomial ($u_0 = s$):

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n.$$

Dealer sends shares $s_i = a(i)$ to \mathcal{P}_i in private, for $i = 1, \dots, \ell$, and broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i , \mathcal{P}_i checks:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \quad (6.1)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.1). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Commitments B_j define **unique** polynomial with coefficients $\log_g B_j$.

Discrete log setting $\langle g \rangle$.

Feldman's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomial ($u_0 = s$):

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n.$$

Dealer sends shares $s_i = a(i)$ to \mathcal{P}_i in private, for $i = 1, \dots, \ell$, and broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i , \mathcal{P}_i checks:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \quad (6.1)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.1). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Commitments B_j define **unique** polynomial with coefficients $\log_g B_j$.



Discrete log setting $\langle g \rangle$.

Feldman's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomial ($u_0 = s$):

$$a(X) = u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n.$$

Dealer sends shares $s_i = a(i)$ to \mathcal{P}_i in private, for $i = 1, \dots, \ell$, and broadcasts commitments $B_j = g^{u_j}$, $0 \leq j < t$. Upon receipt of share s_i , \mathcal{P}_i checks:

$$g^{s_i} = \prod_{j=0}^{t-1} B_j^{i^j}. \quad (6.1)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.1). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Commitments B_j define **unique** polynomial with coefficients $\log_g B_j$.



Security of Feldman VSS

No more than $t - 1$ cheating participants, $t - 1 < \ell/2$.

1. Dealer and participants **bound** to unique polynomial by Eq. (6.1).

2. Secret s **hidden** even given $B_0 = g^s, B_1 = g^{u_1}, \dots, B_{t-1} = g^{u_{t-1}}$.

Reduction proof: given instance of DL problem h successful collusion of participants $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$ would compute $\log_g h$. As follows.

Dealer sets $B_0 = h$, $s_1, \dots, s_{t-1} \in_R \mathbb{Z}_n$, and B_j for $j = 1, \dots, t - 1$ such that Eq. (6.1) holds for $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$, without knowing $s = \log_g h$:

$$B_j = \prod_{k=1}^{t-1} (g^{s_k} / h)^{\gamma_{j,k}}, \quad (6.2)$$

where

$$(\gamma_{j,k}) = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 2 & 2^2 & \dots & 2^{t-1} \\ \vdots & \vdots & & \vdots \\ t-1 & (t-1)^2 & \dots & (t-1)^{t-1} \end{pmatrix}^{-1}$$

Shares for $\mathcal{P}_t, \dots, \mathcal{P}_\ell$ are irrelevant.

Security of Feldman VSS

No more than $t - 1$ cheating participants, $t - 1 < \ell/2$.

1. Dealer and participants **bound** to unique polynomial by Eq. (6.1).
2. Secret s **hidden** even given $B_0 = g^s, B_1 = g^{u_1}, \dots, B_{t-1} = g^{u_{t-1}}$.

Reduction proof: given instance of DL problem h successful collusion of participants $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$ would compute $\log_g h$. As follows.

Dealer sets $B_0 = h$, $s_1, \dots, s_{t-1} \in_R \mathbb{Z}_n$, and B_j for $j = 1, \dots, t - 1$ such that Eq. (6.1) holds for $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$, without knowing $s = \log_g h$:

$$B_j = \prod_{k=1}^{t-1} (g^{s_k} / h)^{\gamma_{j,k}}, \quad (6.2)$$

where

$$(\gamma_{j,k}) = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 2 & 2^2 & \dots & 2^{t-1} \\ \vdots & \vdots & & \vdots \\ t-1 & (t-1)^2 & \dots & (t-1)^{t-1} \end{pmatrix}^{-1}$$

Shares for $\mathcal{P}_t, \dots, \mathcal{P}_\ell$ are irrelevant.

Security of Feldman VSS

No more than $t - 1$ cheating participants, $t - 1 < \ell/2$.

1. Dealer and participants **bound** to unique polynomial by Eq. (6.1).
2. Secret s **hidden** even given $B_0 = g^s, B_1 = g^{u_1}, \dots, B_{t-1} = g^{u_{t-1}}$.

Reduction proof: given instance of DL problem h successful collusion of participants $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$ would compute $\log_g h$. As follows.

Dealer sets $B_0 = h$, $s_1, \dots, s_{t-1} \in_R \mathbb{Z}_n$, and B_j for $j = 1, \dots, t - 1$ such that Eq. (6.1) holds for $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$, without knowing $s = \log_g h$:

$$B_j = \prod_{k=1}^{t-1} (g^{s_k} / h)^{\gamma_{j,k}}, \quad (6.2)$$

where

$$(\gamma_{j,k}) = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 2 & 2^2 & \dots & 2^{t-1} \\ \vdots & \vdots & & \vdots \\ t-1 & (t-1)^2 & \dots & (t-1)^{t-1} \end{pmatrix}^{-1}$$

Shares for $\mathcal{P}_t, \dots, \mathcal{P}_\ell$ are irrelevant.

Security of Feldman VSS

No more than $t - 1$ cheating participants, $t - 1 < \ell/2$.

1. Dealer and participants **bound** to unique polynomial by Eq. (6.1).
2. Secret s **hidden** even given $B_0 = g^s, B_1 = g^{u_1}, \dots, B_{t-1} = g^{u_{t-1}}$.

Reduction proof: given instance of DL problem h successful collusion of participants $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$ would compute $\log_g h$. As follows.

Dealer sets $B_0 = h$, $s_1, \dots, s_{t-1} \in_R \mathbb{Z}_n$, and B_j for $j = 1, \dots, t - 1$ such that Eq. (6.1) holds for $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$, without knowing $s = \log_g h$:

$$B_j = \prod_{k=1}^{t-1} (g^{s_k} / h)^{\gamma_{j,k}}, \quad (6.2)$$

where

$$(\gamma_{j,k}) = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 2 & 2^2 & \dots & 2^{t-1} \\ \vdots & \vdots & & \vdots \\ t-1 & (t-1)^2 & \dots & (t-1)^{t-1} \end{pmatrix}^{-1}$$

Shares for $\mathcal{P}_t, \dots, \mathcal{P}_\ell$ are irrelevant.

Security of Feldman VSS

No more than $t - 1$ cheating participants, $t - 1 < \ell/2$.

1. Dealer and participants **bound** to unique polynomial by Eq. (6.1).
2. Secret s **hidden** even given $B_0 = g^s, B_1 = g^{u_1}, \dots, B_{t-1} = g^{u_{t-1}}$.

Reduction proof: given instance of DL problem h successful collusion of participants $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$ would compute $\log_g h$. As follows.

Dealer sets $B_0 = h$, $s_1, \dots, s_{t-1} \in_R \mathbb{Z}_n$, and B_j for $j = 1, \dots, t - 1$ such that Eq. (6.1) holds for $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$, without knowing $s = \log_g h$:

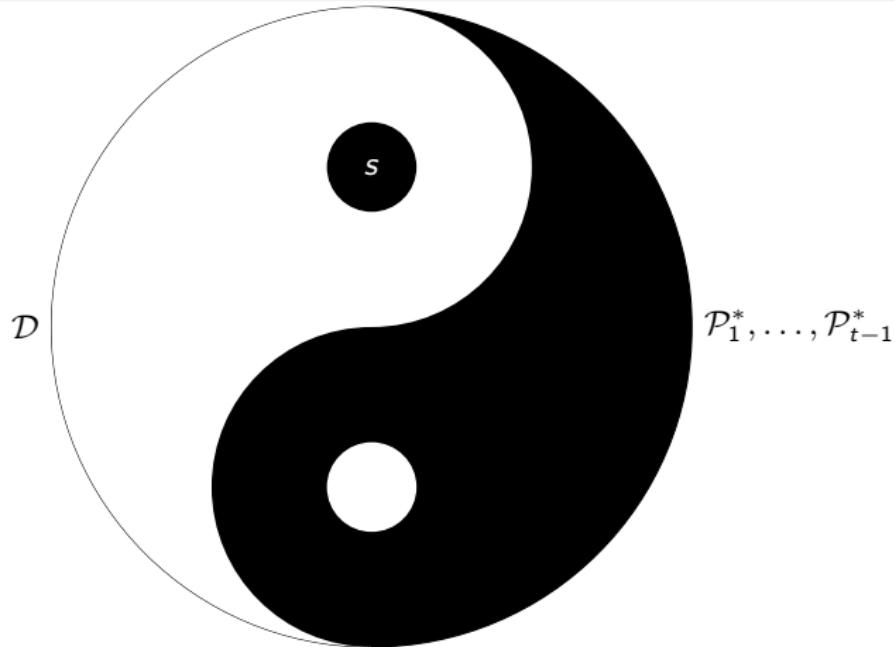
$$B_j = \prod_{k=1}^{t-1} (g^{s_k} / h)^{\gamma_{j,k}}, \quad (6.2)$$

where

$$(\gamma_{j,k}) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 2 & 2^2 & \cdots & 2^{t-1} \\ \vdots & \vdots & & \vdots \\ t-1 & (t-1)^2 & \cdots & (t-1)^{t-1} \end{pmatrix}^{-1}$$

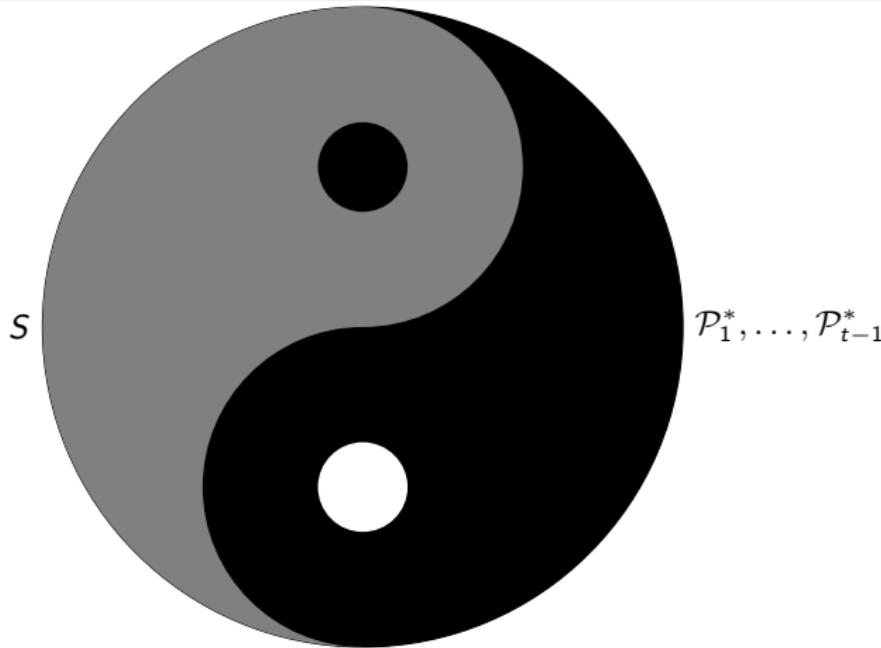
Shares for $\mathcal{P}_t, \dots, \mathcal{P}_\ell$ are irrelevant.

Simulation View



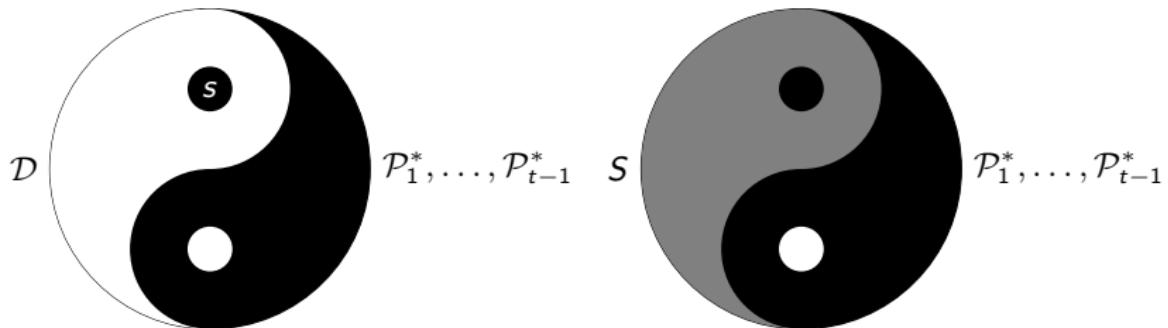
Real protocol runs between dealer \mathcal{D} and participants $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$,
 \mathcal{D} using secret s

Simulation View



Simulated protocol runs between simulator S and participants $\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$,
 S not knowing $s = \log_g B_0$

Simulation View



$\mathcal{P}_1^*, \dots, \mathcal{P}_{t-1}^*$ cannot distinguish between protocol with \mathcal{D} and protocol with S
⇒ protocol is “zero-knowledge” w.r.t. secret s

Exercises

Exercise 6.4

Verify that Eq. (6.1) holds for $1 \leq i < t$ if $B_0 = h$ and B_j , $1 \leq j < t$, are defined by (6.2).

Exercise 6.5

The special case of an (ℓ, ℓ) -threshold scheme for secrets $s \in \mathbb{Z}_n$ can be solved simply by setting the shares as follows: choose $s_i \in_R \mathbb{Z}_n$ for $i = 2, \dots, \ell$ and set $s_1 = (s - \sum_{i=2}^{\ell} s_i) \bmod n$. Extend this basic secret sharing scheme to a Feldman VSS scheme, and provide a security analysis of the resulting VSS scheme.



Discrete log setting $\langle g \rangle$, $h \in_R \langle g \rangle$ such that $\log_g h$ not known.

Pedersen's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomials $a(X), b(X)$ with $u_0 = s$:

$$\begin{aligned} a(X) &= u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n \\ b(X) &= v_0 + v_1 X + \cdots + v_{t-1} X^{t-1}, \quad v_0, \dots, v_{t-1} \in_R \mathbb{Z}_n \end{aligned}$$

Dealer sends shares $s_i = (a(i), b(i))$ to \mathcal{P}_i in private, and broadcasts commitments $C_j = g^{u_j} h^{v_j}$, $0 \leq j < t$.

Upon receipt of $s_i = (s_{i1}, s_{i2})$, \mathcal{P}_i checks:

$$g^{s_{i1}} h^{s_{i2}} = \prod_{j=0}^{t-1} C_j^{i^j}. \quad (6.3)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.3). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Discrete log setting $\langle g \rangle$, $h \in_R \langle g \rangle$ such that $\log_g h$ not known.

Pedersen's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomials $a(X), b(X)$ with $u_0 = s$:

$$\begin{aligned} a(X) &= u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n \\ b(X) &= v_0 + v_1 X + \cdots + v_{t-1} X^{t-1}, \quad v_0, \dots, v_{t-1} \in_R \mathbb{Z}_n \end{aligned}$$

Dealer sends shares $s_i = (a(i), b(i))$ to \mathcal{P}_i in private, and broadcasts commitments $C_j = g^{u_j} h^{v_j}$, $0 \leq j < t$.
 Upon receipt of $s_i = (s_{i1}, s_{i2})$, \mathcal{P}_i checks:

$$g^{s_{i1}} h^{s_{i2}} = \prod_{j=0}^{t-1} C_j^{i^j}. \quad (6.3)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.3). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Discrete log setting $\langle g \rangle$, $h \in_R \langle g \rangle$ such that $\log_g h$ not known.

Pedersen's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomials $a(X), b(X)$ with $u_0 = s$:

$$\begin{aligned} a(X) &= u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n \\ b(X) &= v_0 + v_1 X + \cdots + v_{t-1} X^{t-1}, \quad v_0, \dots, v_{t-1} \in_R \mathbb{Z}_n \end{aligned}$$

Dealer sends shares $s_i = (a(i), b(i))$ to \mathcal{P}_i in private, and broadcasts commitments $C_j = g^{u_j} h^{v_j}$, $0 \leq j < t$.

Upon receipt of $s_i = (s_{i1}, s_{i2})$, \mathcal{P}_i checks:

$$g^{s_{i1}} h^{s_{i2}} = \prod_{j=0}^{t-1} C_j^{i^j}. \quad (6.3)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.3). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Discrete log setting $\langle g \rangle$, $h \in_R \langle g \rangle$ such that $\log_g h$ not known.

Pedersen's (t, ℓ) -threshold VSS scheme

For secret $s \in \mathbb{Z}_n$.

Distribution. Dealer chooses random polynomials $a(X), b(X)$ with $u_0 = s$:

$$\begin{aligned} a(X) &= u_0 + u_1 X + \cdots + u_{t-1} X^{t-1}, \quad u_1, \dots, u_{t-1} \in_R \mathbb{Z}_n \\ b(X) &= v_0 + v_1 X + \cdots + v_{t-1} X^{t-1}, \quad v_0, \dots, v_{t-1} \in_R \mathbb{Z}_n \end{aligned}$$

Dealer sends shares $s_i = (a(i), b(i))$ to \mathcal{P}_i in private, and broadcasts commitments $C_j = g^{u_j} h^{v_j}$, $0 \leq j < t$.

Upon receipt of $s_i = (s_{i1}, s_{i2})$, \mathcal{P}_i checks:

$$g^{s_{i1}} h^{s_{i2}} = \prod_{j=0}^{t-1} C_j^{i^j}. \quad (6.3)$$

Reconstruction. Share s_i contributed by \mathcal{P}_i checked using Eq. (6.3). Secret $s = a(0)$ recovered as in Shamir's scheme from t valid shares.

Security of Pedersen VSS

1. Dealer and participants **bound** to unique polynomial by Eq. (6.3).
Under DL assumption!

Cf. Pedersen commitments are **computationally binding**.

2. Secret s **hidden** information-theoretically,
even given commitments C_0, C_1, \dots, C_{t-1} .

Cf. Pedersen commitments are **information-theoretically hiding**.

Exercise 6.6

See Exercise 6.5. This time extend the basic (ℓ, ℓ) -threshold scheme with shares $s_i \in_R \mathbb{Z}_n$ for $i = 2, \dots, \ell$ and $s_1 = (s - \sum_{i=2}^{\ell} s_i) \bmod n$ to a Pedersen VSS scheme, and provide a security analysis of the resulting VSS scheme.

Security of Pedersen VSS

1. Dealer and participants **bound** to unique polynomial by Eq. (6.3).
Under DL assumption!

Cf. Pedersen commitments are **computationally binding**.

2. Secret s **hidden** information-theoretically,
even given commitments C_0, C_1, \dots, C_{t-1} .

Cf. Pedersen commitments are **information-theoretically hiding**.

Exercise 6.6

See Exercise 6.5. This time extend the basic (ℓ, ℓ) -threshold scheme with shares $s_i \in_R \mathbb{Z}_n$ for $i = 2, \dots, \ell$ and $s_1 = (s - \sum_{i=2}^{\ell} s_i) \bmod n$ to a Pedersen VSS scheme, and provide a security analysis of the resulting VSS scheme.

Security of Pedersen VSS

1. Dealer and participants **bound** to unique polynomial by Eq. (6.3).
Under DL assumption!

Cf. Pedersen commitments are **computationally binding**.

2. Secret s **hidden** information-theoretically,
even given commitments C_0, C_1, \dots, C_{t-1} .

Cf. Pedersen commitments are **information-theoretically hiding**.

Exercise 6.6

See Exercise 6.5. This time extend the basic (ℓ, ℓ) -threshold scheme with shares $s_i \in_R \mathbb{Z}_n$ for $i = 2, \dots, \ell$ and $s_1 = (s - \sum_{i=2}^{\ell} s_i) \bmod n$ to a Pedersen VSS scheme, and provide a security analysis of the resulting VSS scheme.

Definition 6.7 ((t, ℓ)-threshold cryptosystem)

Three components involving parties $\mathcal{P}_1, \dots, \mathcal{P}_\ell$.

Distributed key generation. Protocol between $\mathcal{P}_1, \dots, \mathcal{P}_\ell$ for generating public key h such that \mathcal{P}_i obtains private share x_i (of private key x corresponding to h) and public **verification key** h_i , $1 \leq i \leq \ell$. The protocol depends on t .

Encryption. Algorithm that on input of plaintext M , public key h , outputs ciphertext C of M under public key h .

Threshold decryption. Protocol between t parties $\mathcal{P}_{i_1}, \dots, \mathcal{P}_{i_t}$ that on input of ciphertext C , private shares x_{i_1}, \dots, x_{i_t} , verification keys h_{i_1}, \dots, h_{i_t} , outputs plaintext M .

Exercise 6.8

See Section 2.1.4. Design an (ℓ, ℓ) -threshold ElGamal cryptosystem (for plaintexts in $\langle g \rangle$) with public key $h = \prod_{i=1}^{\ell} h_i$, where $x_i \in \mathbb{Z}_n$ is the private share of \mathcal{P}_i and $h_i = g^{x_i}$ is the corresponding verification key, $1 \leq i \leq \ell$, and discuss its security.



Distributed key generation protocol

- ① Each \mathcal{P}_i picks random polynomial $a_i(X) \in \mathbb{Z}_n[X]$ of degree $< t$, and broadcasts commitment to g^{s_i} , where $s_i = a_i(0)$.
- ② Each \mathcal{P}_i opens its commitment to g^{s_i} . Public key h is set as $h = g^{\sum_{i=1}^{\ell} s_i}$.
- ③ Each \mathcal{P}_i , $1 \leq i \leq \ell$, runs instance of Feldman's VSS scheme, using $s_i \in \mathbb{Z}_n$ as secret value. \mathcal{P}_i plays the role of dealer, and $\mathcal{P}_1, \dots, \mathcal{P}_\ell$ play the role of participants. (Hence, \mathcal{P}_i plays a double role.)
- ④ Let s_{ij} denote the share of s_i as sent by party \mathcal{P}_i to party \mathcal{P}_j , for $1 \leq i, j \leq \ell$. Each party \mathcal{P}_i sums all its received shares s_{ji} to obtain its share $x_i = \sum_{j=1}^{\ell} s_{ji}$ of the private key x . The verification key of party \mathcal{P}_i is defined as $h_i = g^{x_i}$.

Note that $s_{ji} = a_j(i)$. Since $a(X) = \sum_{i=1}^{\ell} a_i(X)$, it follows that $x_i = a(i)$.

Nobody knows x !

Distributed key generation protocol

- ① Each \mathcal{P}_i picks random polynomial $a_i(X) \in \mathbb{Z}_n[X]$ of degree $< t$, and broadcasts commitment to g^{s_i} , where $s_i = a_i(0)$.
- ② Each \mathcal{P}_i opens its commitment to g^{s_i} . Public key h is set as $h = g^{\sum_{i=1}^{\ell} s_i}$.
- ③ Each \mathcal{P}_i , $1 \leq i \leq \ell$, runs instance of Feldman's VSS scheme, using $s_i \in \mathbb{Z}_n$ as secret value. \mathcal{P}_i plays the role of dealer, and $\mathcal{P}_1, \dots, \mathcal{P}_\ell$ play the role of participants. (Hence, \mathcal{P}_i plays a double role.)
- ④ Let s_{ij} denote the share of s_i as sent by party \mathcal{P}_i to party \mathcal{P}_j , for $1 \leq i, j \leq \ell$. Each party \mathcal{P}_i sums all its received shares s_{ji} to obtain its share $x_i = \sum_{j=1}^{\ell} s_{ji}$ of the private key x . The verification key of party \mathcal{P}_i is defined as $h_i = g^{x_i}$.

Note that $s_{ji} = a_j(i)$. Since $a(X) = \sum_{i=1}^{\ell} a_i(X)$, it follows that $x_i = a(i)$.

Nobody knows x !

Threshold decryption protocol

Let $C = (A, B)$ be an ElGamal ciphertext for public key h .

- ① Each \mathcal{P}_i takes A as input and uses its share x_i to produce a value $d_i = A^{x_i}$ along with a Σ -proof showing that $\log_g h_i = \log_A d_i$.
- ② Let Q be a set of t parties who produced correct d_i values. Then plaintext M can be recovered by evaluating:

$$B / \prod_{i \in Q} d_i^{\lambda_{Q,i}} = B / A^x = M,$$

where $\lambda_{Q,i} = \prod_{j \in Q \setminus \{i\}} \frac{j}{j-i}$ denote Lagrange coefficients as in Shamir's scheme.

7.1 Electronic Voting

7.2 Based on Threshold Homomorphic Cryptosystems
7.3 Based on Oblivious Transfer

YES/NO referendum is secure multiparty computation for function

$$f(x_1, \dots, x_\ell) = x_1 + \dots + x_\ell, \text{ for votes } x_1, \dots, x_\ell \in \{0, 1\}.$$

Inputs x_1, \dots, x_ℓ are **private**.

Output $x_1 + \dots + x_\ell$ is **public**.

Security requirements:

- **Eligibility:** only eligible voters can cast vote, at most one vote.
- **Privacy:** assured against any reasonably sized coalition of parties (not including the voter herself).
- **Universal Verifiability:** anyone can check that the election is fair, i.e., published final tally is computed fairly from ballots that were correctly cast.
- **Robustness:** faulty behavior (benign or malicious) of any reasonably sized coalition of participants can be tolerated (e.g., cheating voter can be detected and discarded).



7.1 Electronic Voting

7.2 Based on Threshold Homomorphic Cryptosystems
7.3 Based on Oblivious Transfer

YES/NO referendum is secure multiparty computation for function

$$f(x_1, \dots, x_\ell) = x_1 + \dots + x_\ell, \text{ for votes } x_1, \dots, x_\ell \in \{0, 1\}.$$

Inputs x_1, \dots, x_ℓ are **private**.

Output $x_1 + \dots + x_\ell$ is **public**.

Security requirements:

- **Eligibility:** only eligible voters can cast vote, at most one vote.
- **Privacy:** assured against any reasonably sized coalition of parties (not including the voter herself).
- **Universal Verifiability:** anyone can check that the election is fair, i.e., published final tally is computed fairly from ballots that were correctly cast.
- **Robustness:** faulty behavior (benign or malicious) of any reasonably sized coalition of participants can be tolerated (e.g., cheating voter can be detected and discarded).



Threshold Homomorphic ElGamal Cryptosystem

Threshold ElGamal cryptosystem with \mathbb{Z}_n as plaintext space instead of $\langle g \rangle$.

Homomorphic ElGamal encryption

For plaintext $M \in \mathbb{Z}_n$ and public key h , ciphertext is defined as

$$(A, B) = (g^u, h^u g^M), \quad u \in_R \mathbb{Z}_n.$$

In other words: $M \in \mathbb{Z}_n$ is *encoded* as $g^M \in \langle g \rangle$.

(Additive) homomorphic property

Product of encryptions $(A, B), (A', B')$ of plaintexts M, M' , respectively, is encryption of **sum** of plaintexts $M + M'$:

$$(A, B) * (A', B') = (AA', BB') = (g^{u+u'}, h^{u+u'} g^{M+M'}).$$

Electronic voting scheme

Key generation. Talliers $\mathcal{T}_1, \dots, \mathcal{T}_\ell$ run DKG protocol of (t, ℓ) -threshold ElGamal cryptosystem. Let h denote resulting public key.

Voting. Voter \mathcal{V}_i casts vote $v_i \in \{0, 1\} \simeq \{\text{"no"}, \text{"yes"}\}$ by broadcasting a **ballot** consisting of ElGamal encryption

$(A_i, B_i) = (g^{u_i}, h^{u_i} g^{v_i})$, $u_i \in_R \mathbb{Z}_n$, and non-interactive Σ -proof that (A_i, B_i) is correctly formed.

Tallying. Talliers decrypt $(A, B) = \prod_{i=1}^{\ell'} (A_i, B_i)$ to obtain $g^{\sum_{i=1}^{\ell'} v_i}$, from which $\sum_{i=1}^{\ell'} v_i$ is easily determined using $0 \leq \sum_{i=1}^{\ell'} v_i \leq \ell'$.

If all (A_i, B_i) are correctly formed, homomorphic property yields

$(A, B) = (g^u, h^u g^{\sum_{i=1}^{\ell'} v_i})$ for some $u \in \mathbb{Z}_n$, ensuring validity of final tally.

Exercise 7.1

Provide the Σ -protocol for proving that (A_i, B_i) is correctly formed, and turn it into a non-interactive Σ -proof (see Section 5.4.2 for a similar case).



Exercise 7.2 (Boardroom election scheme)

Voter \mathcal{V}_i has a public key $h_i = g^{x_i}$, where $x_i \in_R \mathbb{Z}_n$ is \mathcal{V}_i 's private key. Let $H_i = h_\ell \prod_{j=1}^{i-1} h_j$, for $1 \leq i \leq \ell$.

First, voter \mathcal{V}_ℓ publishes the following encryption of its vote $v_\ell \in \{0, 1\}$:

$$(A_\ell, B_\ell) = (g^{u_\ell}, H_\ell^{u_\ell} g^{v_\ell}), \quad u_\ell \in_R \mathbb{Z}_n.$$

Next, for $i = \ell - 1, \dots, 1$ (in this order), voter \mathcal{V}_i publishes the following encryption of its vote $v_i \in \{0, 1\}$:

$$(A_i, B_i) = (A_{i+1} g^{u_i}, B_{i+1} A_{i+1}^{-x_i} H_i^{u_i} g^{v_i}), \quad u_i \in_R \mathbb{Z}_n.$$

Finally, voter \mathcal{V}_ℓ publishes $\log_g B_1 A_1^{-x_\ell}$.

Let $t_i = \sum_{j=i}^\ell v_j$, for $1 \leq i \leq \ell$. (i) Prove by induction on i that (A_i, B_i) is ElGamal encryption of g^{t_i} under public key H_i . Hence, \mathcal{V}_ℓ publishes $\sum_{j=1}^\ell v_j$ at the end of the protocol. (ii) Show how $\mathcal{V}_\ell, \mathcal{V}_1, \dots, \mathcal{V}_{i-1}$ for any i , $2 \leq i \leq \ell$, are jointly able to decrypt (A_i, B_i) , hence able to determine intermediate election result t_i . (iii) Describe the relations to be proved in each protocol step to show that the voter's output is formed correctly.

Matching without Embarrassments

Alice's rule



for "yes"



for "no"

separator



Bob's rule



for "yes"



for "no"

Matching without Embarrassments

Alice's rule



for "yes"



for "no"

separator



Bob's rule



for "yes" for "no"



Suppose Alice thinks "yes"



separator



Suppose Bob thinks "yes"



Matching without Embarrassments

Alice's rule



for "yes"



for "no"

separator



Bob's rule



for "yes"

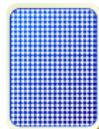


for "no"

Suppose Alice thinks "yes"



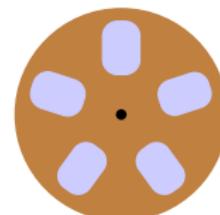
separator



Suppose Bob thinks "yes"

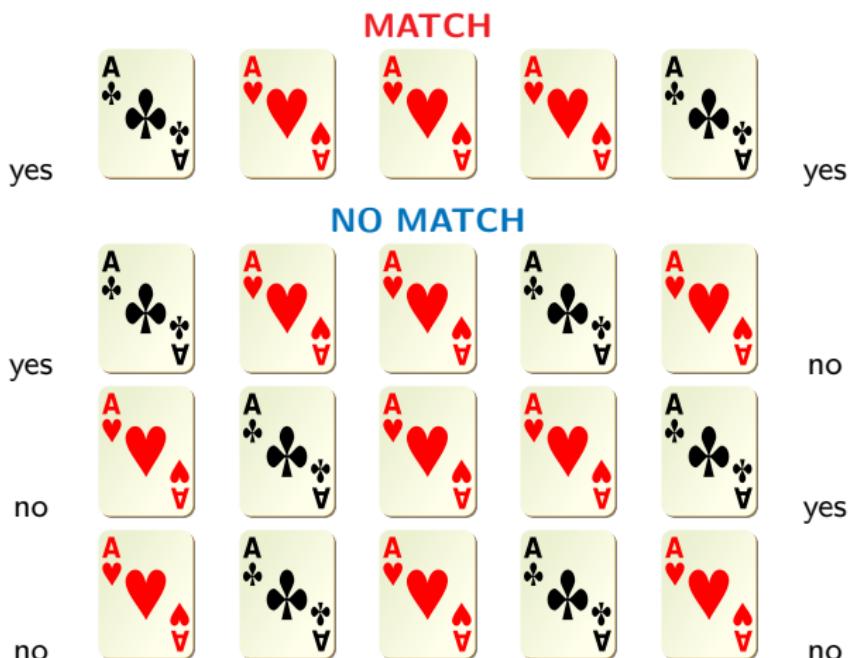


Alice and Bob make a random cut:



and open the cards ...

Five-Card Trick (Bert den Boer, Eurocrypt 1989)



“NO MATCH” cases are indistinguishable!

If you put “no” you don’t find out the other’s preference.

Check out Tom Verhoeff's wonderful [3D printable smiley design](#)

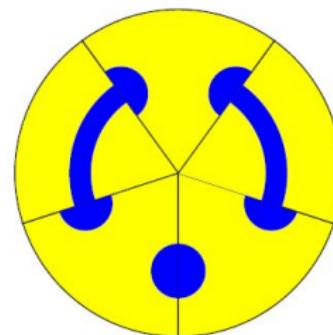


Figure 7.1 (Matching without embarrassments)

x	y	xy
0	0	0
1	0	0
0	1	0
1	1	1

 \approx

Alice	Bob	match?
no	no	-
yes	no	-
no	yes	-
yes	yes	♡

Check out Tom Verhoeff's wonderful [3D printable smiley design](#)

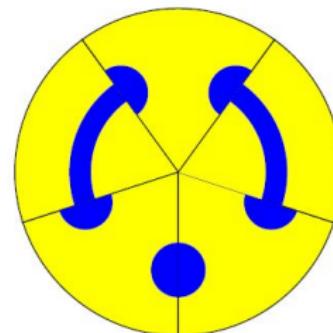


Figure 7.1 (Matching without embarrassments)

x	y	xy
0	0	0
1	0	0
0	1	0
1	1	1

\approx

Alice	Bob	match?
no	no	-
yes	no	-
no	yes	-
yes	yes	♡

\mathcal{A} and \mathcal{B} set up $(2, 2)$ -threshold homomorphic ElGamal with public key h .

Secure multiplication: private inputs x, y , public output xy

Party \mathcal{A}
 $(x \in \{0, 1\})$

$$\begin{aligned} u &\in_R \mathbb{Z}_n \\ A &\leftarrow g^u \end{aligned}$$

$$B \leftarrow h^u g^x$$

$$(A, B) \longrightarrow$$

Party \mathcal{B}
 $(y \in \{0, 1\})$

$$\begin{aligned} v &\in_R \mathbb{Z}_n \\ C &\leftarrow g^v A^y \\ D &\leftarrow h^v B^y \end{aligned}$$

$$(C, D) \longleftarrow$$

$$\xleftarrow{\text{jointly decrypt } (C, D)} \xrightarrow{\quad}$$

output xy

output xy

Note: $(C, D) = (g^{v+uy}, h^{v+uy} g^{xy})$.

\mathcal{A} and \mathcal{B} set up $(2, 2)$ -threshold homomorphic ElGamal with public key h .

Secure multiplication: private inputs x, y , public output xy

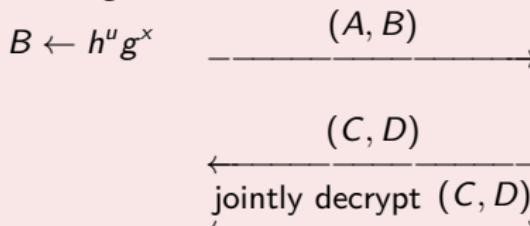
Party \mathcal{A}
 $(x \in \{0, 1\})$

$$\begin{aligned} u &\in_R \mathbb{Z}_n \\ A &\leftarrow g^u \end{aligned}$$

$$B \leftarrow h^u g^x$$

Party \mathcal{B}
 $(y \in \{0, 1\})$

$$\begin{aligned} v &\in_R \mathbb{Z}_n \\ C &\leftarrow g^v A^y \\ D &\leftarrow h^v B^y \end{aligned}$$



Note: $(C, D) = (g^{v+uy}, h^{v+uy} g^{xy})$.

\mathcal{A} and \mathcal{B} set up $(2, 2)$ -threshold homomorphic ElGamal with public key h .

Secure multiplication: private inputs x, y , public output xy

Party \mathcal{A}
 $(x \in \{0, 1\})$

$$\begin{aligned} u &\in_R \mathbb{Z}_n \\ A &\leftarrow g^u \end{aligned}$$

$$B \leftarrow h^u g^x$$

$$(A, B) \longrightarrow$$

Party \mathcal{B}
 $(y \in \{0, 1\})$

$$\begin{aligned} v &\in_R \mathbb{Z}_n \\ C &\leftarrow g^v A^y \\ D &\leftarrow h^v B^y \end{aligned}$$

$$(C, D)$$

$$\xleftarrow{\text{jointly decrypt } (C, D)}$$

output xy

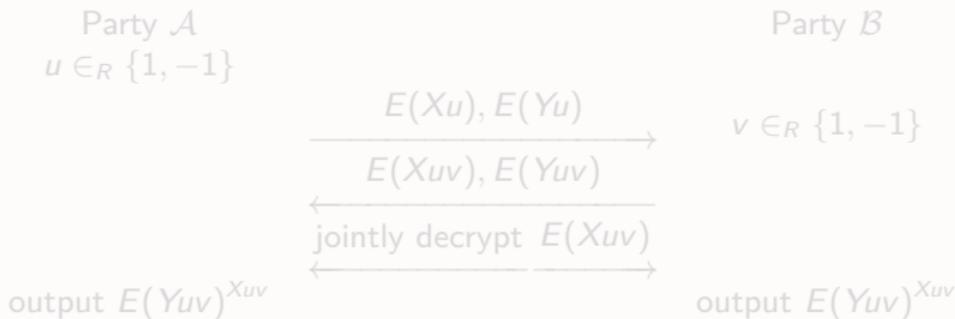
output xy

Note: $(C, D) = (g^{v+uy}, h^{v+uy} g^{xy})$.

Let $X = (-1)^x$ and $Y = (-1)^y$.

Encryptions $E(X)$, $E(Y)$ common input to parties \mathcal{A} and \mathcal{B} .

Secure multiplication: input $E(X)$, $E(Y)$, output $E(XY)$



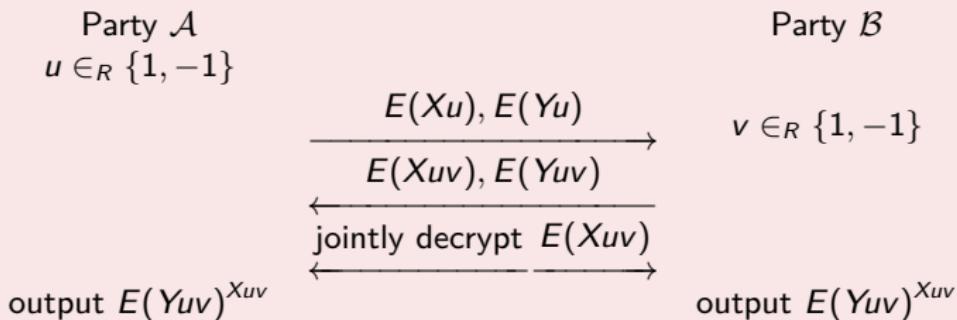
Note:

- $E(Yuv)^{Xuv} = E(YuvXuv) = E(XYv^2u^2) = E(XY)$.
- Xuv as decrypted during protocol is statistically independent of X :
 - from \mathcal{A} 's point of view: \mathcal{A} does not know $v \in_R \{1, -1\}$
 - from \mathcal{B} 's point of view: \mathcal{B} does not know $u \in_R \{1, -1\}$

Let $X = (-1)^x$ and $Y = (-1)^y$.

Encryptions $E(X)$, $E(Y)$ common input to parties \mathcal{A} and \mathcal{B} .

Secure multiplication: input $E(X)$, $E(Y)$, output $E(XY)$



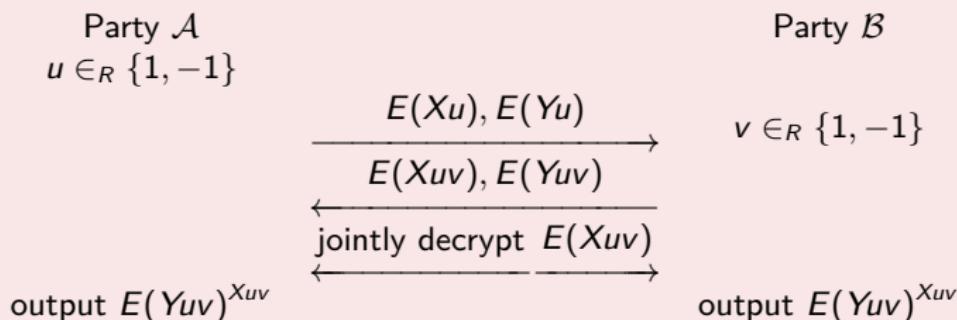
Note:

- $E(Yuv)^{Xuv} = E(YuvXuv) = E(XYu^2v^2) = E(XY)$.
- Xuv as decrypted during protocol is statistically independent of X :
 - from \mathcal{A} 's point of view: \mathcal{A} does not know $v \in_R \{1, -1\}$
 - from \mathcal{B} 's point of view: \mathcal{B} does not know $u \in_R \{1, -1\}$

Let $X = (-1)^x$ and $Y = (-1)^y$.

Encryptions $E(X)$, $E(Y)$ common input to parties \mathcal{A} and \mathcal{B} .

Secure multiplication: input $E(X)$, $E(Y)$, output $E(XY)$



Note:

- $E(Yuv)^{Xuv} = E(YuvXuv) = E(XYu^2v^2) = E(XY)$.
- Xuv as decrypted during protocol is statistically independent of X :
 - from \mathcal{A} 's point of view: \mathcal{A} does not know $v \in_R \{1, -1\}$
 - from \mathcal{B} 's point of view: \mathcal{B} does not know $u \in_R \{1, -1\}$

“Raw” OT

Figure 7.2 (Rabin OT protocol)



Both cases probability 50%.

Receiver knows whether it gets b or \perp .

Sender does not know whether bit b was transferred successfully or not.

Strange functionality of OT sufficiently powerful to construct secure computation for *any* computable function: OT is complete for secure computation.

“Raw” OT

Figure 7.2 (Rabin OT protocol)



Both cases probability 50%.

Receiver knows whether it gets b or \perp .

Sender does not know whether bit b was transferred successfully or not.

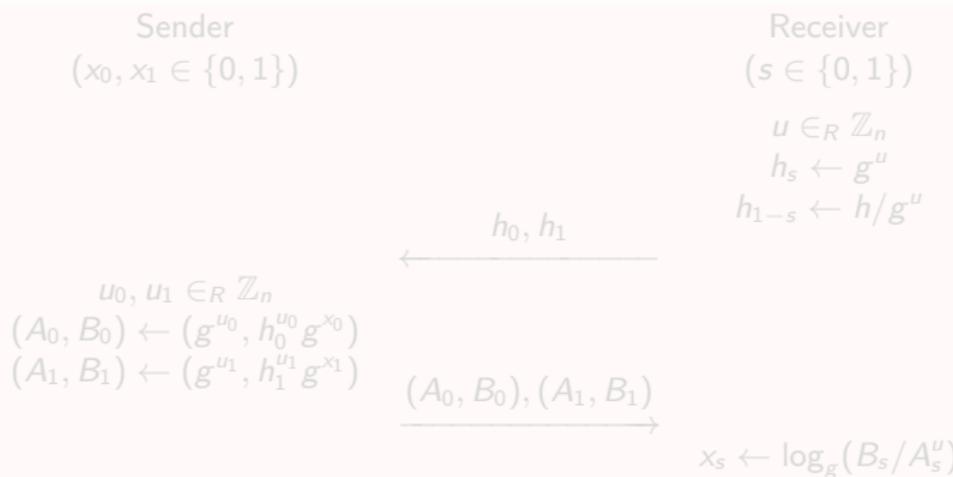
Strange functionality of OT sufficiently powerful to construct secure computation for *any* computable function: OT is complete for secure computation.

1-out-of-2 OT

Sender holds two data bits x_0, x_1 , receiver holds one selection bit s .

Functionality: $OT(x_0, x_1; s) = x_s$.

Figure 7.3 ($\binom{2}{1}$ -OT protocol)

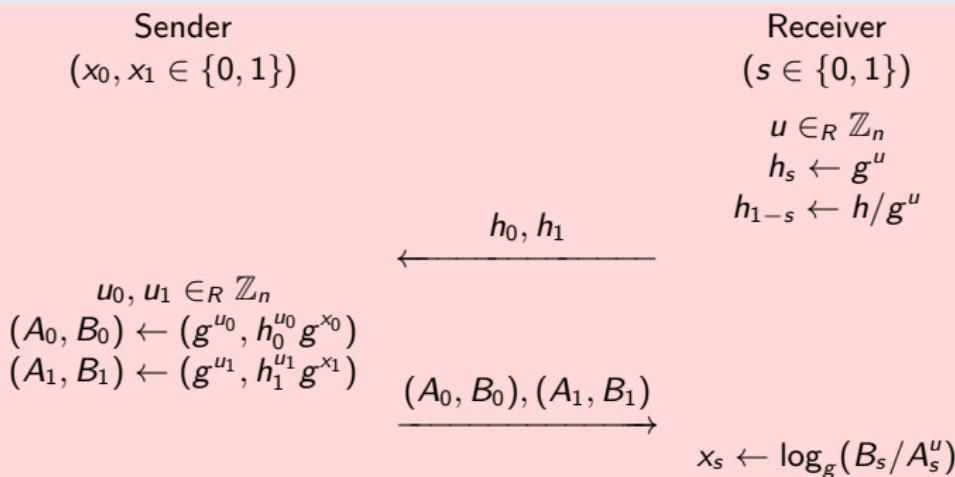


1-out-of-2 OT

Sender holds two data bits x_0, x_1 , receiver holds one selection bit s .

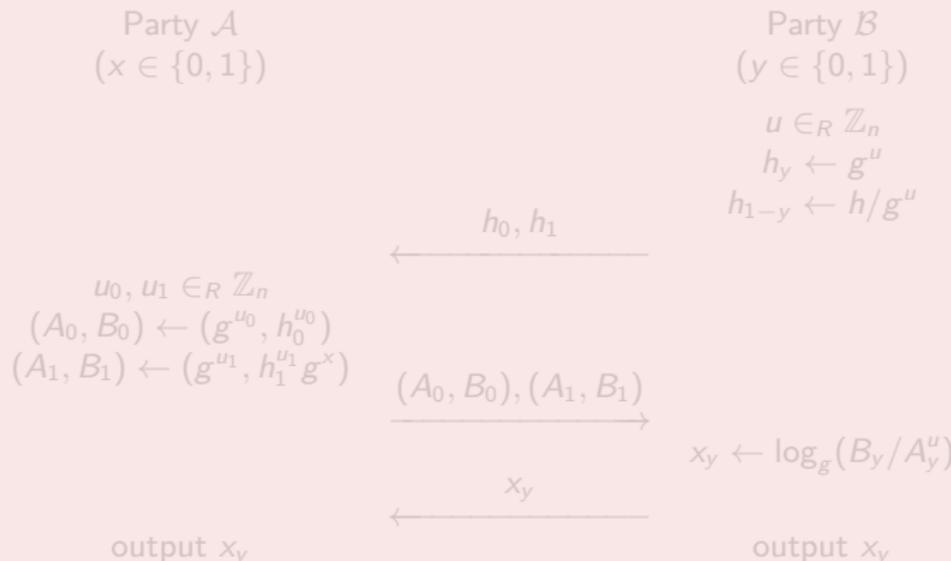
Functionality: $OT(x_0, x_1; s) = x_s$.

Figure 7.3 $\binom{2}{1}$ -OT protocol)



Property: $OT(0, x; y) = xy$

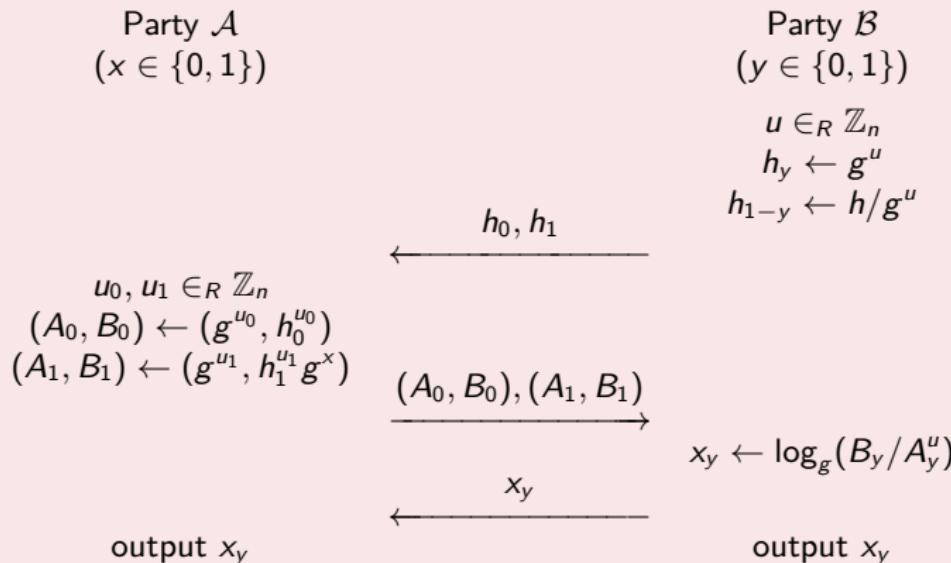
Secure multiplication: private inputs x, y , public output xy



Note: $x_y = \log_g(B_y / A_y^u) = xy$.

Property: $OT(0, x; y) = xy$

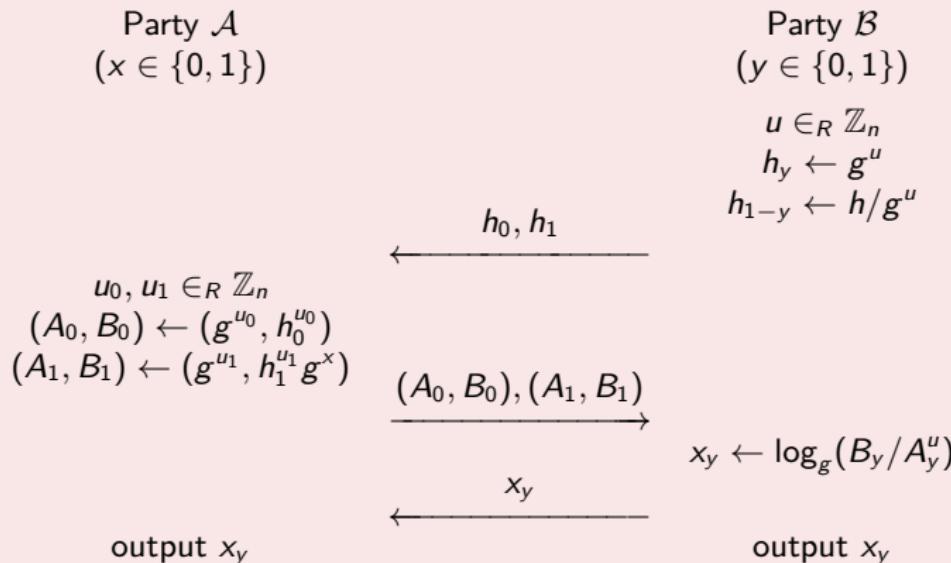
Secure multiplication: private inputs x, y , public output xy



Note: $x_y = \log_g(B_y/A_y^u) = xy$.

Property: $OT(0, x; y) = xy$

Secure multiplication: private inputs x, y , public output xy



Note: $x_y = \log_g(B_y/A_y^u) = xy$.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = OT(u_b, x_b \oplus u_b; y_a)$

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b **uniform and independent** of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = OT(u_b, x_b \oplus u_b; y_a)$

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b **uniform and independent** of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = OT(u_b, x_b \oplus u_b; y_a)$

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b uniform and independent of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = OT(u_b, x_b \oplus u_b; y_a) = u_b(1 - y_a) \oplus (x_b \oplus u_b)y_a$

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b **uniform and independent** of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = \dots = u_b(1 - y_a) \oplus (x_b \oplus u_b)y_a = u_b \oplus u_b y_a \oplus x_b y_a \oplus u_b y_a$

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b uniform and independent of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = \dots = u_b \oplus u_b y_a \oplus x_b y_a \oplus u_b y_a = u_b \oplus x_b y_a$

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b uniform and independent of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = u_b \oplus x_b y_a$

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b uniform and independent of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = u_b \oplus x_b y_a$ and $v_b = u_a \oplus x_a y_b$.

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b uniform and independent of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = u_b \oplus x_b y_a$ and $v_b = u_a \oplus x_a y_b$.

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b uniform and independent of other values due to u_b, u_a , resp.

\mathcal{A} and \mathcal{B} hold $(2, 2)$ -threshold secret shares of bits x, y , cf. Exercise 6.2.

Secure multiplication: input $x = x_a \oplus x_b, y = y_a \oplus y_b$, output $xy = z_a \oplus z_b$

Party \mathcal{A}
 $(x_a, y_a \in \{0, 1\})$
 $u_a \in_R \{0, 1\}$

Party \mathcal{B}
 $(x_b, y_b \in \{0, 1\})$
 $u_b \in_R \{0, 1\}$

$$\frac{OT(u_a, x_a \oplus u_a; y_b)}{OT(u_b, x_b \oplus u_b; y_a)}$$

$$v_a \leftarrow OT(u_b, x_b \oplus u_b; y_a) \\ z_a \leftarrow x_a y_a \oplus u_a \oplus v_a$$

$$v_b \leftarrow OT(u_a, x_a \oplus u_a; y_b) \\ z_b \leftarrow x_b y_b \oplus u_b \oplus v_b$$

Property: $OT(x_0, x_1; s) = x_s = x_0(1 - s) \oplus x_1 s$.

Therefore: $v_a = u_b \oplus x_b y_a$ and $v_b = u_a \oplus x_a y_b$.

So $z = z_a \oplus z_b = xy$ follows from: $\begin{cases} z_a = x_a y_a \oplus u_a \oplus u_b \oplus x_b y_a \\ z_b = x_b y_b \oplus u_b \oplus u_a \oplus x_a y_b \end{cases}$

Shares z_a, z_b **uniform and independent** of other values due to u_b, u_a , resp.

Definition 8.1 (Blind signature scheme)

Key generation. Algorithm that on input of security parameter k , generates key pair (sk, pk) consisting of private key sk and public key pk .

Signature generation. Two-party protocol between **signer** \mathcal{S} and **receiver** \mathcal{R} with public key pk as common input. Private input of \mathcal{S} is private key sk , and private input of \mathcal{R} is message M . At the end of the protocol, \mathcal{R} obtains signature S on M as *private output*.

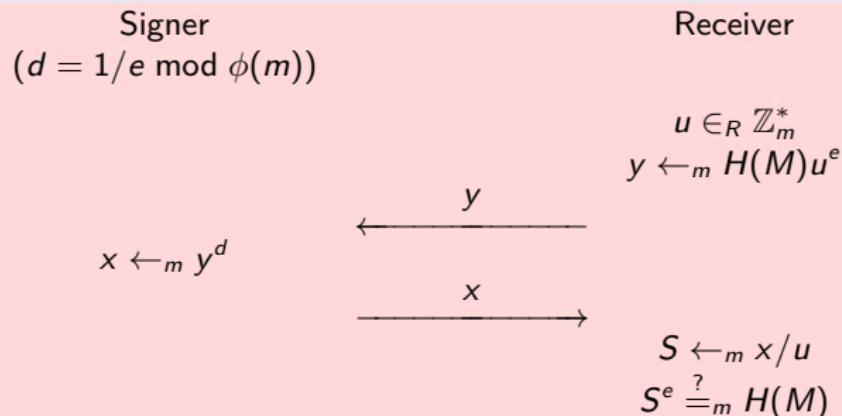
Signature verification. Algorithm that on input of message M , public key pk , and signature S , determines whether S is a valid signature on M w.r.t. public key pk .

- unforgeability
- unlinkability

Chaum's Blind Signatures

RSA setting: RSA modulus m , public exponent e .

Figure 8.1 (Chaum's blind signature protocol)

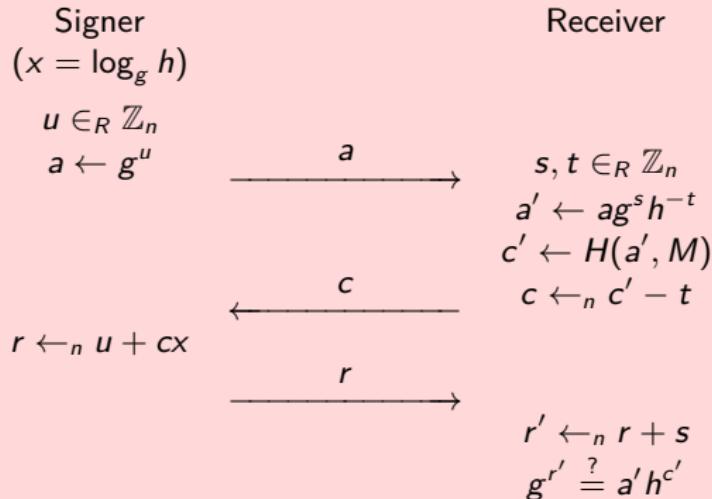


Receiver obtains RSA signature S on message M :

$$S^e = (x/u)^e = (y^{1/e}/u)^e = y/u^e = H(M) \text{ mod } m$$

Perfect unlinkability: (x, y) is uniformly random, independent of (M, S) .

Figure 8.2 (Schnorr-based blind signature protocol)



Receiver obtains Schnorr signature $S = (c', r')$ on message M :

$$c' = H(a', M) = H(ag^s h^{-t}, M) = H(g^{r+s} h^{-c-t}, M) = H(g^{r'} h^{-c'}, M).$$

Perfect unlinkability: (c, r) is uniformly random, independent of (c', r') .

Exercises

Exercise 8.2

Consider Chaum's blind signature protocol (Figure 8.1) for a fixed message M . Show that the pair (x, y) is distributed uniformly random by proving that

$$\Pr[(x, y) = (x_0, y_0)] = 1/\phi(m)$$

for any $x_0, y_0 \in \mathbb{Z}_m^$ satisfying $y_0 = x_0^e \bmod m$.*

Exercise 8.3

Consider the Schnorr-based blind signature protocol (Figure 8.2) for a fixed message M . Show that the triples (a, c, r) , (a', c', r') are distributed uniformly random and independent of each other by proving that

$$\Pr[(a; c; r) = (a_0; c_0; r_0) \wedge (a'; c'; r') = (a'_0; c'_0; r'_0)] = 1/n^3$$

for any $(a_0; c_0; r_0)$, $(a'_0; c'_0; r'_0)$ s.t. $g^{r_0} = a_0 h^{c_0}$, $c'_0 = H(a'_0, M)$, and $g^{r'_0} = a'_0 h^{c'_0}$.

A.1 Spell Check

Spell Check

authentication	authentification
ciphertext	cipher text
cryptanalysis	cryptoanalysis, crypto analysis
cryptosystem	crypto system
identification	indentification
plaintext	plain text