

Information Security

Андрій Степанович Олійник

Київський національний університет імені Тараса Шевченка
goo.gl/U5VYGV

- ① Introduction
- ② Information security
- ③ Cryptographic Primitives
- ④ RSA
- ⑤ Block and Stream Ciphers
- ⑥ Hash functions and MAC
- ⑦ Digital Signatures, Certificates and PKI
- ⑧ Key Exchange
- ⑨ TLS
- ⑩ Elliptic Curves
- ⑪ Bilinear Pairing
- ⑫ Identity-based Encryption
- ⑬ Functional Encryption
- ⑭ Homomorphic Encryption
- ⑮ Post Quantum Cryptography

Introduction

Outline

Purpose of the course overview state-of-the-art of information security and cryptology.

Include theoretical and practical aspects.

Theory modern cryptographic approaches to information security.

Practice real-world cryptographic standards and protocol.

Requirements basic knowledge of number theory, groups, rings, finite fields, linear algebra.

Recommended basic notions and examples in cryptology.

Topics to consider

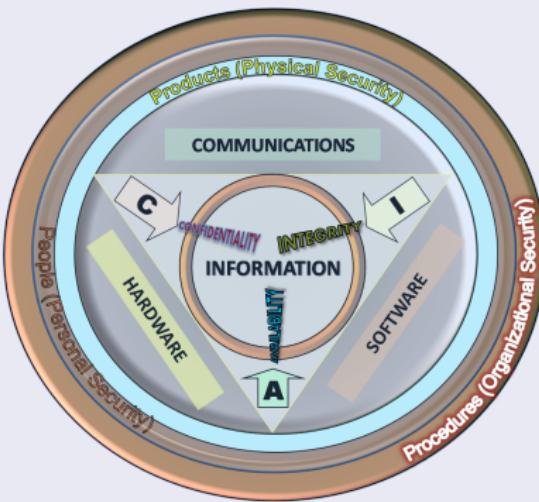
- Information security.
- Cryptographic primitives.
- Provable security.
- Block and stream ciphers.
- White-box cryptography.
- Hash functions.
- Digital signatures.
- Key management.
- Certificates and PKI.
- SSL/TLS and OpenSSL.
- Elliptic curves.
- Distributed ledgers.
- Bilinear pairing.
- Identity based encryption.
- Attribute based encryption.
- Functional encryption.
- Post-quantum cryptography.
- Lattices.
- Homomorphic cryptography.
- Searchable encryption.
- Oblivious transfer.
- Multiparty computations.
- Zero-knowledge protocols.
- Online voting.

Grading and Action items

- Final grade consists of the semester grade (up to 60) and the exam grade (up to 40)
- The semester grade consists of the project grade (presentation, documentation, demo, source) (up to 40) and mid-term exam (up to 20)
- Fill the form goo.gl/DgdQSB
- Create teams
- Choose topic for the project

Information security

Main goals of information security



CIA triad

- **Confidentiality:** information is not made available or disclosed to unauthorized individuals, entities, or processes
- **Integrity:** maintaining and assuring the accuracy and completeness of data over its entire life-cycle
- **Availability:** information MUST be available when it is needed

Risks and Threats

- Computer crime
- Vulnerability
- Zero-day attacks
- Eavesdropping
- Spoofing
- Exploits
- Trojans
- Viruses and worms
- Denial of service
- Malware
- Ransomware
- Payloads
- Rootkits
- Keyloggers
- PHA

Backdoor

Definition

An application that allows the execution of unwanted, potentially harmful remote-controlled operations on a device that would place the app into one of the other malware categories if executed automatically.

In general, the backdoor is more a description of how potentially harmful operation can happen on a device and is therefore not completely aligned with PHA categories like billing fraud or commercial spyware apps.

Spyware

Definition

Any application that transmits sensitive information off the device without user consent and does not display a persistent notification that this is happening.

Examples are:

- contact list
- photos or other files not owned by the application
- content from user email
- call log
- SMS log
- web history or browser bookmarks of the default browser
- information from the /data/ directories of other applications

Denial of service

Definition

An application that, without the knowledge of the user, executes a denial-of-service attack or is a part of a distributed denial-of- service attack against other systems and resources.

This can happen by sending a high volume of HTTP requests to produce excessive load on remote servers.

Hostile downloader

Definition

An application that is not in itself potentially harmful, but downloads other potentially harmful apps.

For example, a gaming app that does not contain malicious code, but persistently displays a misleading Security Update link that installs harmful apps.

Mobile billing fraud

Definition

An application that charges the user in an intentionally misleading way.

Mobile billing fraud is divided into SMS fraud, Call fraud, and Toll fraud based on the type of fraud being committed.

Phishing

Definition

An application that pretends to come from a trustworthy source, requests a user's authentication credentials and/or billing information, and sends the data to a third party.

This category also applies to apps that intercept the transmission of user credentials in transit. Common targets of phishing include banking credentials, credit card numbers, or online account credentials for social networks and games.

Mobile unwanted software (MUwS)

Definition

Any application that collects at least one of the following without user consent:

- Information about installed applications
- Information about third-party accounts
- Names of files on the device

This includes collecting the actual list of installed applications as well as partial information like information about currently active apps.

Privilege escalation

Definition

An application that compromises the integrity of the system by breaking the application sandbox, or changing or disabling access to core security-related functions.

Examples include:

- An app that violates the Android permissions model, or steals credentials (such as OAuth tokens) from other apps.
- An app that prevents its own removal by abusing device administrator APIs.
- An app that disables SELinux.

Privilege escalation apps that root devices without user permission are classified as rooting apps.

Ransomware

Definition

An application that takes partial or extensive control of a device or data on a device and demands payment to release control.

Some ransomware apps encrypt data on the device and demand payment to decrypt data and/or leverage the device administrator features so that the app can't be removed by the typical user. Examples include:

- Ransomware that locks a user out of their device and demands money to restore user control.
- Ransomware that encrypts data on the phone and demands payment, ostensibly to decrypt data again.
- Ransomware that leverages device policy manager features and cannot be removed by the user.

Rooting

Definition

A privilege escalation app that roots the device.

There is a difference between malicious rooting apps and non-malicious rooting apps.

Non-malicious rooting apps let the user know in advance that they are going to root the device and they do not execute other potentially harmful actions that apply to other PHA categories.

Malicious rooting apps do not inform the user that they will root the device, or they inform the user about the rooting in advance but also execute other actions that apply to other PHA categories.

Spam

Definition

An application that sends unsolicited commercial messages to the users contact list or uses the device as an email spam relay.

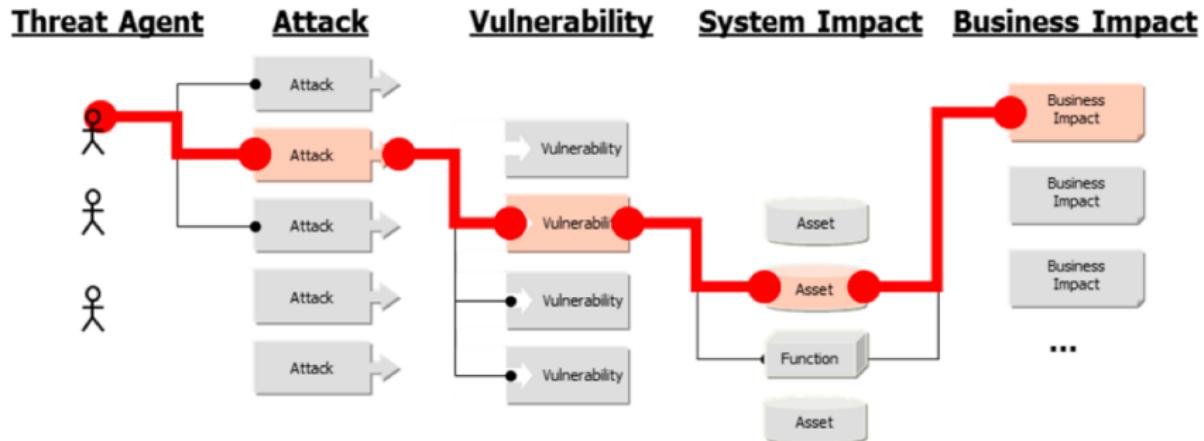
Trojan

Definition

An application that appears to be benign, such as a game that claims only to be a game, and performs undesirable actions against the user.

This classification is usually used in combination with other categories of harmfulness. A trojan will have an innocuous app component and a hidden harmful component. For example, a tic-tac-toe game that, in the background and without the knowledge of the user, sends premium SMS messages from the users device

Threat in action

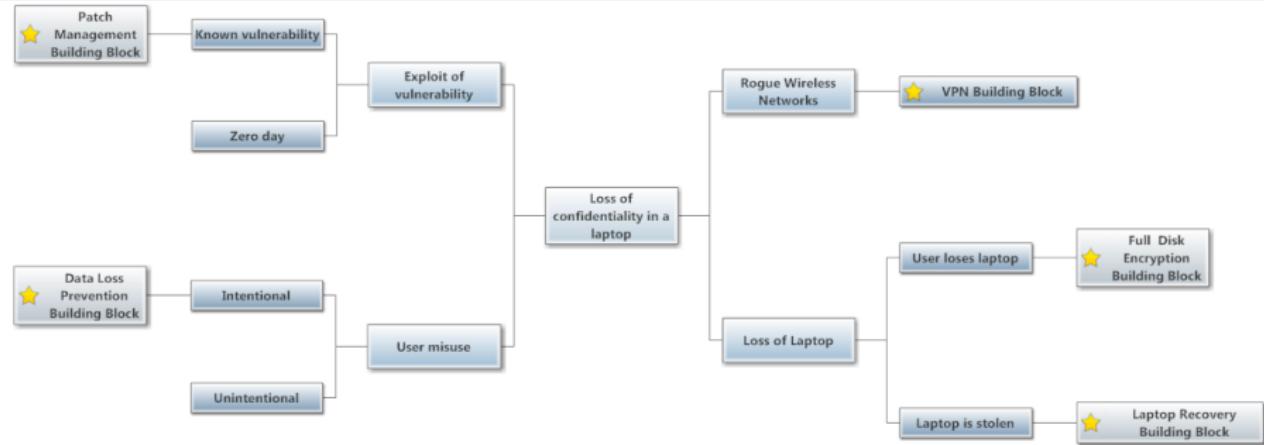


- ① system susceptibility
- ② attacker access
- ③ attacker capability to exploit

Methods of defense

- Access control
- Application security
- Antivirus software
- Secure coding
- Security by design
- Secure operating systems
- Firewall
- Authentication
- Multi-factor authentication
- Two-factor authentication
- Authorization
- Data-centric security
- Intrusion detection system
- Intrusion prevention system
- Mobile secure gateway

Threat modeling



Identification, enumeration and prioritization of potential threats from a hypothetical attacker's point of view

See the other [examples](#)

Common Vulnerabilities and Exposures

[View All CVEs](#)

CVE-ID

CVE-2017-6074 [Learn more at National Vulnerability Database \(NVD\)](#)

- Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings

Description

The dccp_rcv_state_process function in net/dccp/input.c in the Linux kernel through 4.9.11 mishandles DCCP_PKT_REQUEST packet data structures in the LISTEN state, which allows local users to obtain root privileges or cause a denial of service (double free) via an application that makes an IPV6_RECVPKTINFO setsockopt system call.

References

Note: [References](#) are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MLIST:[oss-security] 20170222 Linux kernel: CVE-2017-6074: DCCP double-free vulnerability (local root)
- URL:<http://www.openwall.com/lists/oss-security/2017/02/22/3>
- CONFIRM:<https://github.com/torvalds/linux/commit/Sedabca9d4cff7f1f2b68f0bac55ef99d9798ba4>

Date Entry Created

20170217 Disclaimer: The entry creation date may reflect when the CVE-ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Phase (Legacy)

Assigned (20170217)

Votes (Legacy)

Comments (Legacy)

Proposed (Legacy)

N/A

This is an entry on the [CVE list](#), which standardizes names for security problems.

SEARCH CVE USING KEYWORDS:

You can also search by reference using the [CVE Reference Maps](#).

For More Information: cve@mitre.org

CVE online database cve.mitre.org

InfoSec in general

Prevents unauthorized

- access
- use
- disclosure
- disruption
- modification
- inspection
- recording
- destruction

of information.

...however...

The only reason (at least a widely-used strategy) to buy/use secure solutions is **FUD**,
where **FUD** stands for fear, uncertainty and doubt.

Remarks on InfoSec

- Increasing **Security** decreases **Usability**
 - Password
 - PIN
 - Encryption
 - Time-out
- There is no and will be no **total, complete, comprehensive** security solutions
- Each security solution MUST be rigorously verified before and after implementation

Cryptographic Primitives

Crypto vocabulary

- plaintext
- ciphertext
- encryption
- encryption algorithm
- decryption
- decryption algorithm
- secret (private) key
- public key
- cryptographic system (cipher)
- block cipher
- stream cipher
- key space
- brute-force attack
- codebook attack
- algebraic attack
- time-memory tradeoff
- cryptography
- cryptanalysis
- cryptology

Private-Key Crypto

Two parties (e.g. Alice and Bob) wish to communicate secretly.

An adversary (e.g. Eve) observes the communication.

It is assumed that Alice and Bob share a secret key such that

- the key is not known to Eve
- this key used for both encryption and decryption

Syntax of Secret-key Cryptography

Space of keys \mathcal{K} , space of plaintexts \mathcal{M} , space of ciphertexts \mathcal{C} .

Three algorithms **Keygen**, **Enc**, **Dec**.

- Initialization algorithm **Keygen** outputs public parameters and a secret key $sk \in \mathcal{K}$.
- Encryption algorithm **Enc** takes the key sk and a plaintext $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.
- Decryption algorithm **Dec** takes the key sk and a ciphertext $c \in \mathcal{C}$ and outputs a plaintext $m \in \mathcal{M}$.

Correctness requirement

$$\text{Dec}_{\text{sk}}(\text{Enc}_{\text{sk}}(m)) = m.$$

Kerckhoffs' principle: all algorithms **Keygen**, **Enc**, **Dec** are public and the only secret is the key sk .

Historical ciphers

- The Caesar cipher
- The Vigenère cipher
- One-time pad cipher

Public-key Crypto

Introduced by Whitfield Diffie and Martin Hellman in 1976 in the paper
[“New directions of cryptography”](#).

Each partie (Alice, Bob) has its own public key for encryption and secret key for decryption .

Everybody can encrypt but only the owner of the secret key can decrypt.

Syntax of Public-key Cryptography

Spaces of secret keys \mathcal{SK} and public keys \mathcal{PK} , space of plaintexts \mathcal{M} , space of ciphertexts \mathcal{C} .

Again three (probabilistic) algorithms **Keygen**, **Enc**, **Dec**.

- Initialization algorithm **Keygen** outputs public parameters and a secret key $sk \in \mathcal{SK}$ and a public key $pk \in \mathcal{PK}$.
- Encryption algorithm **Enc** takes the public key pk and a plaintext $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.
- Decryption algorithm **Dec** takes the secret key sk and a ciphertext $c \in \mathcal{C}$ and outputs a plaintext $m \in \mathcal{M}$.

Correctness requirement

$$\Pr[\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m] = 1.$$

- All algorithms **Keygen**, **Enc**, **Dec** and key pk are public and the only secret is the secret key sk .

Crypto design

While constructing a cryptosystem answer the questions.

- ① What is the functionality of the system?
- ② How do we expect the system to get attacked? Can we formalize a security definition?
- ③ What are the building blocks? Are they used in an intuitive way?
- ④ What assumptions can be made about the security of the building blocks?
- ⑤ Can we reduce the security of the system to that of the building blocks?

Crypto security

The precise definition of security depends on

- ① the accepted notion of the break
 - recover the secret key
 - recover the plaintext
 - recover a part of the plaintext
 - learn anything meaningful about the plaintext
- ② the computational ability of the adversary
 - probabilistic polynomial time (PPT)
 - computationally unbounded

Crypto security

③ type of attack of the adversary

- known-ciphertext attack
(Eve observes a challenge ciphertext c^* only)
- known-plaintext attack
(Eve additionally learns some pairs $(m, \mathbf{Enc}(m))$ and observes a challenge ciphertext c^*)
- chosen-plaintext attack (CPA)
(Eve additionally learns some pairs $(m, \mathbf{Enc}(m))$ for plaintexts m of her choice and observes a challenge ciphertext c^*)
- chosen-ciphertext attack (CCA)
(Eve additionally learns some pairs $(m, \mathbf{Enc}(m))$ for plaintexts m of her choice, $(c, \mathbf{Dec}(c))$ for ciphertexts c of her choice and observes a challenge ciphertext c^*).

Crypto security

A cryptographic scheme for a given task is secure if no adversary of a specified power can achieve a specified break.

A scheme is secure if every probabilistic polynomial-time (PPT) adversary succeeds in breaking the scheme with only negligible probability.

Negligible functions

Definition

A function $\nu : \mathbb{N} \rightarrow \mathbb{R}^+$ is **negligible** if for every polynomial $p(\cdot)$ there exists a number N such that for all $n > N$ it holds that

$$\nu(n) < \frac{1}{p(n)}.$$

Functions $\frac{1}{2^n}$, $\frac{1}{2\sqrt{n}}$, $\frac{n^{2014}}{2^{\log^2 n}}$ are negligible.

The sum of negligible functions is negligible.

The product of a negligible function by a polynomial is negligible.

Indistinguishability

The CPA **indistinguishability experiment** $\text{PubK}_{\mathcal{A}, \Pi}^{cpa}(n)$

- ① $\text{Keygen}(1^n)$ is run to obtain keys (pk, sk) .
- ② Adversary \mathcal{A} is given pk as well as oracle access to $\mathbf{Enc}_{pk}(\Delta)$. The adversary outputs a pair of messages m_0, m_1 of the same length from the plaintext space associated with pk .
- ③ A random bit $b \in \{0, 1\}$ is chosen, and then a ciphertext $c \leftarrow \mathbf{Enc}_{pk}(m_b)$ is computed and given to \mathcal{A} . It is called the challenge ciphertext.
- ④ \mathcal{A} continues to have access to $\mathbf{Enc}_{pk}(\Delta)$, and outputs a bit b' .
- ⑤ The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

CPA security

Definition

A public-key encryption scheme $\Pi = (\mathbf{Keygen}, \mathbf{Enc}, \mathbf{Dec})$ has indistinguishable encryptions under a chosen-plaintext attack (or is **CPA secure**) if for all probabilistic polynomial-time adversaries \mathcal{A} there exists a negligible function ν such that

$$\Pr[\text{PubK}_{\mathcal{A}, \Pi}^{cpa}(n) = 1] \leq \frac{1}{2} + \nu(n).$$

Proof of Security by Reduction

Fix some efficient adversary \mathcal{A} attacking Π . Denote this adversary's success probability by $\varepsilon(n)$.

Construct an efficient algorithm \mathcal{A}' , called the **reduction** that attempts to solve computationally hard problem X using adversary \mathcal{A} as a sub-routine. If \mathcal{A} succeeds in breaking the instance of Π that is being simulated by \mathcal{A}' , this should allow \mathcal{A}' to solve the instance $x \in X$ it was given, at least with inverse polynomial probability $1/p(n)$.

If $\varepsilon(n)$ is not negligible, then \mathcal{A}' solves problem X with non-negligible probability $\varepsilon(n)/p(n)$. Since \mathcal{A}' is efficient, and runs the PPT adversary \mathcal{A} as a sub-routine, this implies an efficient algorithm solving X with non-negligible probability, contradicting the initial hardness assumption about X .

RSA

Invented by Ron Rivest, Adi Shamir, and Leonard Adleman in [Rivest, R.; Shamir, A.; Adleman, L. \(1978\). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM 21 \(2\): 120–126.](#)

For details of RSA see the standard [PKCS #1: RSA Cryptography Standard](#).

The textbook RSA is described as follows

RSA key generation

Keygen

- ① Generate two different primes p and q (Miller-Rabin primality test)
- ② Calculate the modulus $N = p \cdot q$ (multiplication of long integers)
- ③ Calculate $\varphi(N) = (p - 1) \cdot (q - 1)$
- ④ Select for public exponent an integer e such that $1 < e < \varphi(N)$ and $GCD(\varphi(N), e) = 1$
- ⑤ Calculate the private exponent d such that

$$d = e^{-1} \pmod{\varphi(N)}, \text{ i.e. } de = 1 \pmod{\varphi(N)}$$

(Euclidean algorithm)

Then the public key pk is (N, e) and the secret key sk is (N, d) .
It is widely used to choose e as 3, 17 or 65 537.

RSA encryption

Enc

- ① Take a plaintext m which is a positive integer (strictly less than the public modulus N and) relatively prime to N
- ② Using the public key $pk = (N, e)$ calculate the ciphertext c as

$$c = m^e \pmod{N}$$

RSA decryption

Dec

- ① Take a ciphertext c which is a positive integer (strictly less than the public modulus N and) relatively prime to N
- ② Using the secret key $sk = (N, d)$ calculate the plaintext m as

$$m = c^d \pmod{N}$$

RSA correctness

Theorem (Euler)

Let N be a positive integer. Then

$$a^{\varphi(N)} = 1 \pmod{N}$$

for each integer a relatively prime to N .

Since $\varphi(N) = (p-1)(q-1)$ and $ed = 1 \pmod{\varphi(N)}$ for some integer s it holds $ed - s(p-1)(q-1) = 1$. Then by Euler's theorem we obtain

$$c^d = (m^e)^d = m^{ed} = m^{1+s(p-1)(q-1)} = m \cdot (m^{(p-1)(q-1)})^s = m \cdot 1^s = m.$$

RSA security

The breaking RSA is the **RSA** problem, i.e.

Given N (that equals $p \cdot q$), e and y relatively prime to N , compute an x such that $x^e = y \pmod{N}$

The security assumption the RSA cryptosystem is based on is the following **FACTORING** problem

A positive integer N such that $N = pq$ for “big” primes p and q is given.
Find p and q .

This problem is believed computationally hard for primes of bit length at least 2048 (see [Recommendation for Key Management, Special Publication 800-57 Part 1 Rev. 3, NIST, 07/2012](#))

The **RSA** problem is no harder than the **FACTORING** problem.

RSA baby example

Let $p = 11$, $q = 23$.

Assume that $e = 3$.

Then $N = p \cdot q = 253$, $\varphi(N) = (p - 1)(q - 1) = 220$, and $d = 147$ (why?).

To encrypt $m = 00111001$ with $pk = (N, e) = (253, 3)$ look at m as at number (how?) 57 (relatively prime to 253). Then compute $c = 57^3 \pmod{253} = 250$.

To decrypt $c = 250$ compute $m = 250^{147} \pmod{253} = 57$.

To play with real-world examples one can use e.g. [SageMath](#) or [OpenSSL](#)

```

p = random_prime(2^1024); p
102489813835669196445590980294565628361485583332737808882262013311259965409021075055503283713090576532951711061847918757662943131309098396302811410588072156025049254569827635784793862043074544506453512501
321878382161441068716477066479578598740173413889189981013169941707995473011056299427701382795096643869

q = random_prime(2^1024); q
1593609785812231039601359428443037619297267955194464052237810741311821214158254784982892407343328431905618744050424933176844526029664360190246644718916285796270257487885468807051533138291234900231225768840
20174650901368525815056533995562134767865085485901496791407397590512984881835938608126950336032726279

n = p*q; n
16332877274596221510976403888166032088940701042885009291051396402256002675456716583689571888984721246893189688043963187863569411538620326309724255897346012234010238294566400184479597283915854917965425
78921438308740644251925229941363415178802559937949788955031642580938659350162183876320338896927493240786631379715115080809083675180852887745423151061645095574395087732832853289680061136574954725546935883
8458107293365689832749618191999975042284072786493738110446398293804489568757843008158808130266873946795858438355199807356362799119583859894516668057125396286858745868835889270898978275788022583220533451

phi = (p-1)*(q-1); phi
16332877274596221510976403888166032088940701042885009291051396402256002675456716583689571888984721246893189688043963187863569411538620326309724255897346012234010238294566400184479597283915854917965425
78921438308740644251925229941363415178802559937949788955031642580938659350162183876320338896927493240786631379715115080809083675180852887745423151061645095574395087732832853289680061136574954725546935883
275735561763714868767432520946541163873384597542137581513965583850846732438842673848643392587592660315596095250569013504893203926843837134777561056189244735961482847710575440674192167051698452091163384

e = random_prime(phi); e
15123582605693721535318345110662956701346053146535035662711771573841189739755822426591593568559174133774759458607935548329505508254407060262201793496184354408597276346689372514928551563575801961983614481
895817678455644615330612173678406410888141114317203292047723275523037841776409382720823677817221476511648788449854681776113898168822644514970696474734265605905817713278659180427812654936968275769846727313
335857838861224832132610229683434052925618387979100062176664164925989243158774518966816319351438858212539768384882480059801015609660024302018184317800340377219335908023488830232653336084054270753876605293

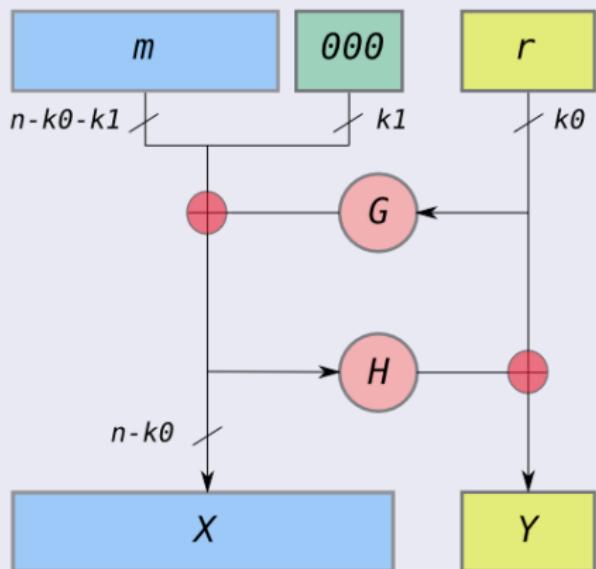
d = xgcd(e, phi)[1]; d
1

plaintext=1234567
ciphertext=power_mod(plaintext,e,n);ciphertext
15576414400511594183094898420384433662251701280599187247343468276509545806345174745236353640198242826772985928483374446595293417945030864673652539786792147168406099684204103059771897575894321267827520547
2696411295452877676714791371354748057416420665407659515873717438713488005020841370962174613245123536046875784883204016023741263051012058050784513873218191601595820110114703246676136137998704610258009552
50599356595547182663028155888181050838691671334865140841990369091372137355479479416809172863455255510862364168378736507606821217410605348836868076870124740800248487103514098459056572263193382437677645261

power_mod(ciphertext,d,n)
1234567

```

RSA-OAEP



Notation

- n — the number of bits in the RSA modulus
- k_0, k_1 — fixed integers
- m — plaintext, $(n - k_0 - k_1)$ -bit string
- r — randomly generated k_0 -bit string
- G, H — cryptographic hash functions
- \oplus — XOR operation
- $X||Y$ — ciphertext

Block and Stream Ciphers

AES

- See [standard FIPS 197](#) for detailed description.
- AES is a particular case of the Rijndael cipher (the authors Vincent Rijmen, Joan Daemen).
- As any other block cipher it can be used as a stream cipher (e.g. in CBC (cipher-block chaining) mode).

AES selection process

- September 12, 1997: the NIST publicly calls for nominees for the new AES
- 1st AES conference, August 20-23, 1998
 - (15 algorithms are candidates for becoming AES)
- Public Review of the algorithms
- 2nd AES conference, March 22-23, 1999
 - (presentation, analysis and testing)
- August 9, 1999: the 5 finalists are announced
 - (**MARS, RC6, RINJDAEL, SERPENT, TWOFISH**)
- Public Review
- 3rd AES conference, April 13-14, 2000
 - (presentation, analysis and testing)

AES selection process

- October 2, 2000: the winner is chosen: RINJDAEL
- February 28, 2001: publication of a Draft by *Federal Information Processing Standard (FIPS)*
- Public Review of 90 days
- Proposal to the *Secretary of Commerce* for approval
- Publication on the *Federal Register*, December 6, 2001,
 - Effective starting from May 26, 2002

Pronunciation: Reign Dahl, Rain Doll, Rhine Dahl

Requirements for AES

- In the selection process, NIST asked for:
 - A block cipher
 - Key length: 128, 192, or 256 bit
 - Block length: 128 bit
 - Possible implementation on smart-cards
 - Royalty-free
- NIST platform used to test candidate cipher algorithms:
 - PC IBM-compatible, Pentium Pro 200 MHz, 64 MB RAM, WINDOWS 95
 - Borland C++ 5.0 compiler, and Java Development Kit (JDK) 1.1
- NIST selection of the winning algorithm based on:
 - Security
 - Efficient implementation *both* in hardware and software
 - Code length and memory utilization

Documentation produced by candidates

- Algorithm Description
- Analysis of the algorithm (advantages/disadvantages)
- Estimation of the computation efficiency
- Algorithm analysis with respect to the best known attacks
(e.g. with known or chosen plaintext)
- Implementation in ANSI C
- Optimized implementation both in ANSI C and Java

Finalists and candidates for AES

RIJNDAEL Joan Daemen, Vincent Rijmen

MARS IBM

RC6 RSA Laboratories

SERPENT R. Anderson, E. Biham, L. Knudsen

TWOFISH B.Schneier, J.Kelsey, D.Whiting, D.Wagner, C.Hall, N.Ferguson

CAST-256 Entrust Technologies, INC.

CRYPTON Future System, INC.

DEAL R. Outerbridge, L.Knudsen

DFC CNRS

E2 Nippon Telegraph and Telephone Corp.

FROG TecApro Internacional S.A.

HPC L.Brown, J.Pieprzyk, J.Seberry

LOKI97 L.Brown, J.Pieprzyk, J.Seberry

MAGENTA Deutsche Telekom AG

SAFER+ Cylink Corp.

AES: Rijndael

- It is not a Feistel cipher.
 - It works in parallel over the whole input block.
 - Designed to be efficient both in hardware and software across a variety of platforms.
 - It's a block cipher which works iteratively
 - Block size: 128 bit (but also 192 or 256 bit)
 - Key length: 128, 192, or 256 bit
 - Number of rounds: 10, 12 or 14
 - Key scheduling: 44, 52 or 60 subkeys having length = 32 bit
- Each round (except the last one) is a uniform and parallel composition of 4 steps
- **SubBytes** (byte-by-byte substitution using an S-box)
 - **ShiftRows** (a permutation, which cyclically shifts the last three rows in the State)
 - **MixColumns** (substitution that uses Galois Fields, *corps de Galois*, GF(2^8) arithmetic)
 - **AddRound key** (bit-by-bit XOR with an expanded key)

AES Parameters

	Key Length (N_k words)	Block Size (N_b words)	Number of Rounds (N_r)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

1 word = 32 bit

AES Keys

- With 128 bit: $2^{128} = 3.4 \times 10^{38}$ possible keys
 - A PC that tries 2^{55} keys per second needs 149.000 billion years to break AES
- Con 192 bit: $2^{192} = 6.2 \times 10^{57}$ possible keys
 - ...
- Con 256 bit: $2^{256} = 1.1 \times 10^{77}$ possible keys
 - ...

Probably AES will stay secure for at least 20 years

Key and Block

➤ Key with variable length (128,192, 256 bit)

- Represented with a **matrix (array) of bytes** with 4 rows and N_k columns, $N_k = \text{key length} / 32$
 - key of 128 bits= 16 bytes $\rightarrow N_k=4$
 - key of 192 bits= 24 bytes $\rightarrow N_k=6$
 - key of 256 bits= 32 bytes $\rightarrow N_k=8$

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$

➤ Block of length 128 bits=16 bytes

- Represented with a **matrix (array) of bytes** with 4 rows and N_b columns, $N_b = \text{block length} / 32$
 - Block of 128 bits= 16 bytes $\rightarrow N_b=4$

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

in =input

10

State

- Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the **State**
 - 4 rows, each containing Nb bytes
 - Nb columns, constituted by 32-bit words
 - $S_{r,c}$ denotes the byte in row r and column c

➤ The array of bytes in input is copied in the **State** matrix

$$S_{r,c} \leftarrow \text{in}$$

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

➤ At the end, the **State matrix** is copied in the output matrix

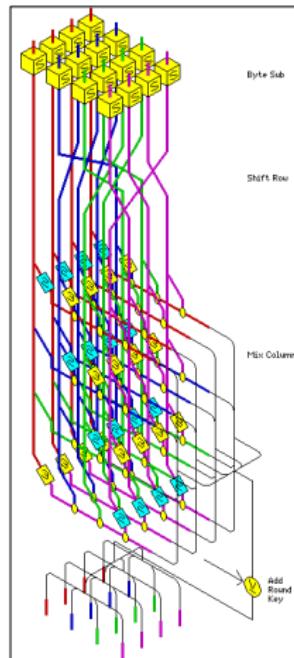
$$\text{out} \leftarrow S_{r,c}$$

Rijndael Design

- Operations performed on State (4 rows of bytes).
- The 128 bit key is expanded as an array of 44 entries of 32 bits words; 4 distinct words serve as a round key for each round; key schedule relies on the S-box
- Algorithms composed of three layers
 - Linear Diffusion
 - Non-linear Diffusion
 - Key Mixing

Rijandael: High-Level Description

-
- State = X
1. AddRoundKey(State, Key₀)
 2. for r = 1 to (Nr - 1)
 - a. SubBytes(State, S-box)
 - b. ShiftRows(State)
 - c. MixColumns(State)
 - d. AddRoundKey(State, Key_r)
 - end for
 1. SubBytes(State, S-box)
 2. ShiftRows(State)
 3. AddRoundKey(State, Key_{Nr})
- Y = State

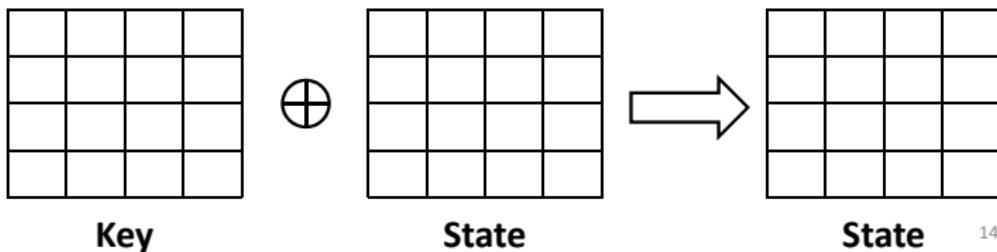


AddRound Key

- State is represented as follows (16 bytes):

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

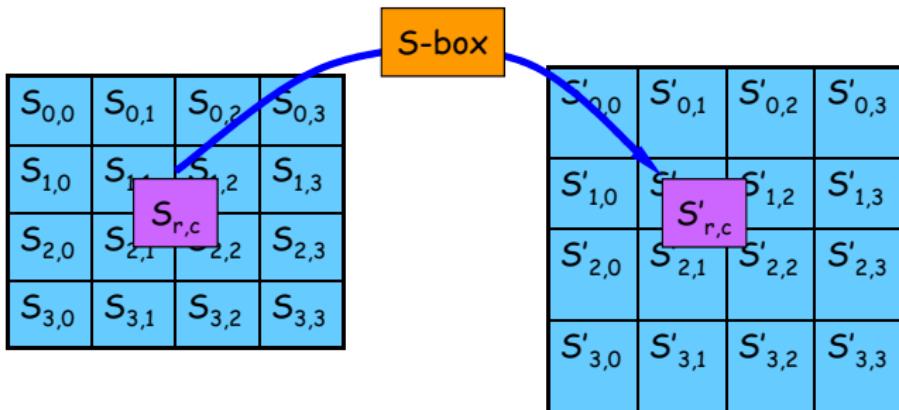
- AddRoundKey(State, Key):



SubBytes Transformation

Bytes are transformed using a *non-linear S-box*

- $S'_{r,c} \leftarrow \text{S-box}(S_{r,c})$



SubBytes

- Byte substitution using a non-linear (but *invertible*) S-Box (independently on each byte).
- S-box is represented as a 16x16 array, rows and columns indexed by hexadecimal bits
- 8 bytes replaced as follows: 8 bytes define a hexadecimal number \mathbf{rc} , then $s_{r,c} = \text{binary}(\text{S-box}(r, c))$
- How is AES S-box different from DES S-boxes?
 - Only one S-box
 - S-boxes based on modular arithmetic with polynomials, can be defined algebraically
 - Easy to analyze, prove attacks fail

Rijandael S-box Table

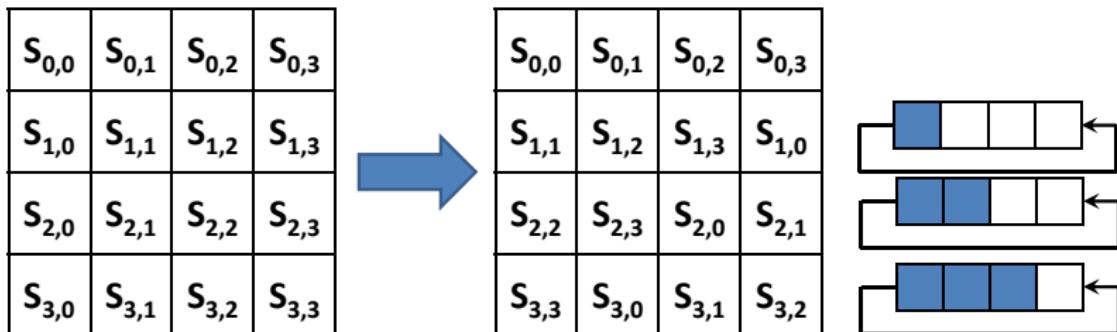
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	3	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Example: hexa 53 is replaced with hexa ED

(The first 4 bits in the byte(the first hexadecimal value, hence) individuate the row,
the last 4 bits individuate the column)

ShiftRows

- Circular Left Shift of a number of bytes equal to the row number



MixColumns

- Interpret each column as a vector of length 4.
- Each column of State is replaced by another column obtained by multiplying that column with a matrix in a particular field (Galois Field).

MixColumns Transformation

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} \leftarrow \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Multiply mod x^4+1 with $a(x)$
 $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$

MixColumns()

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$
	$S'_{0,c}$	$S'_{1,c}$	$S'_{2,c}$

$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,0}$	$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$
$S'_{2,0}$	$S'_{2,1}$	$S'_{2,2}$	$S'_{2,3}$
$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$	$S'_{3,3}$
	$S'_{0,c}$	$S'_{1,c}$	$S'_{2,c}$

Bytes in columns are combined linearly

Decryption

- The decryption algorithm is not identical with the encryption algorithm, but uses the same key schedule.
- There is also a way of implementing the decryption with an algorithm that is equivalent to the encryption algorithm (each operation replaced with its inverse), however, in this case, the key schedule must be changed.

Rijandel Cryptanalysis

- Resistant to linear and differential cryptanalysis
 - Academic break on weaker version of the cipher, 9 rounds.
 - Requires 2^{224} work and 2^{85} chosen *related-key plaintexts*.
 - Attack not practical.

Kalyna

- Kalyna was adopted as the national encryption standard of Ukraine in 2015 (standard DSTU 7624:2014) after holding Ukrainian national cryptographic competition.
- Detailed description at [eprint](#), including test vectors
 - [Reference implementation](#)
 - Implemented in [cppcrypto](#)
 - [Bruce Schneier on Kalyna](#)

Outline

- Second generation of block ciphers in the post-Soviet states
- The new Ukrainian block cipher Kalyna
- Performance comparison with other ciphers
- Other sections of the Ukrainian national standard DSTU 7624:2014
- Conclusions

The block cipher GOST 28147-89

Advantages

- a well known and researched cipher, adopted as the national standard in 1990
- acceptable encryption speed
- appropriate for lightweight cryptography
- "good" S-boxes provide practical strength

Disadvantages

- theoretically broken
- huge classes of weak keys
- special S-boxes allow practical ciphertext-only attacks
- significantly slower performance in comparison to modern block ciphers like AES

Replacements for GOST 28147-89 in Belarus

Belarus: STB 34.101.31-2011

- the block length is 128 bits; the key length is 128, 192 or 256 bits
- 8-round Feistel network with a Lai-Massey scheme
- a single byte S-box with good cryptographic properties
- no key schedule like in GOST
- no cryptanalytical attacks better than exhaustive search are known
- faster than GOST, but slower than AES

Replacements for GOST 28147-89 in Russia

Russia: GOST R 34.12-2015

- the block cipher Magma
 - GOST 28147-89 with fixed substitutions
- the block cipher Kuznyechik
 - the block length is 128 bits; the key length is 256 bits
 - 9 rounds of Rijndael-like transformation
 - single byte S-box (common with "Stribog")
 - non-circulant MDS matrix of 16x16 size over $GF(2^8)$ (different from "Stribog")
 - key schedule based on a Feistel network and involves round transformations
 - no cryptanalytical attacks better than exhaustive search are known
 - faster than GOST, slower* than AES

Ukraine: DSTU 7624:2014

- normal, high and ultra high security level
 - the block and key length 128, 256 and 512 bits
- Rijndael-like SPN structure
- four different S-boxes (not CCZ-equivalent) with optimized cryptographic properties
- 8x8 MDS matrix over $GF(2^8)$
- a new construction of Key Schedule based on the round function
- faster than GOST, faster* than AES

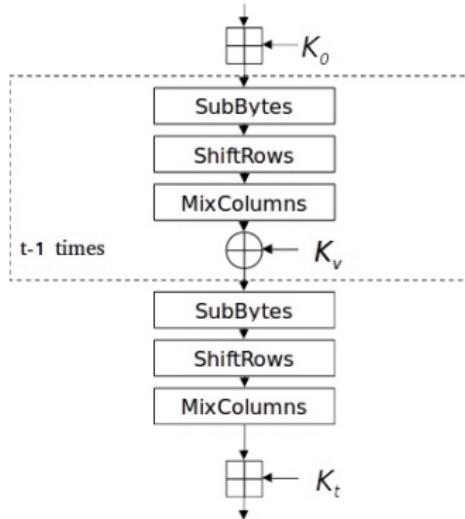
Kalyna: supported block and key lengths

$k \backslash l$	128	256	512	c
128	10	14	-	2
256	-	14	18	4
512	-	-	18	8

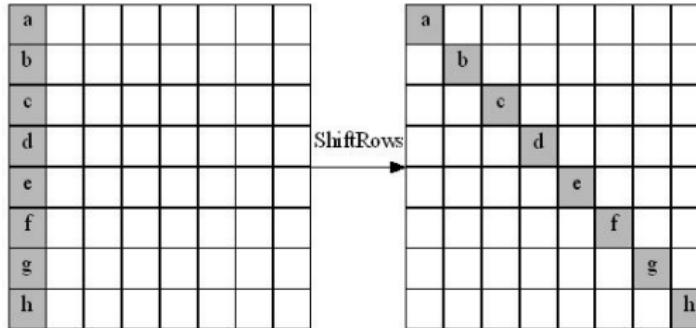
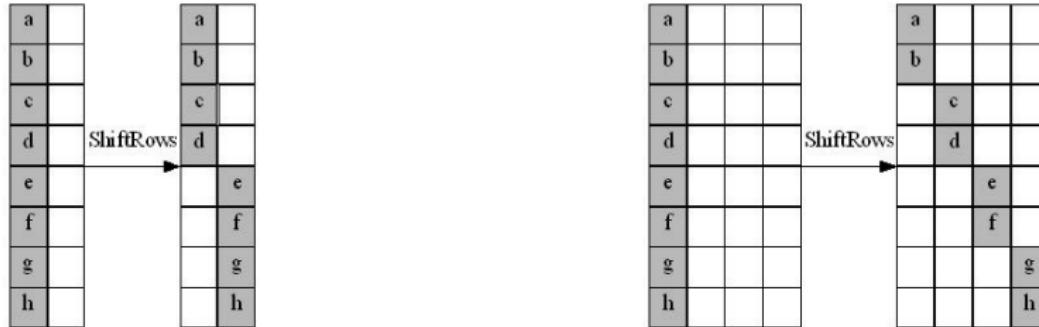
l : the block size; k : the key length;
 t : the number of rounds; c : the number of columns

Kalyna: high-level structure

$$T_{I,k}^{(K)} = \eta_I^{(K_t)} \circ \psi_I \circ \tau_I \circ \pi'_I \circ \prod_{\nu=1}^{t-1} (\kappa_I^{(K_\nu)} \circ \psi_I \circ \tau_I \circ \pi'_I) \circ \eta_I^{(K_0)}$$



Kalyna: permutation of elements



Kalyna: properties of S-boxes

Property	S-box			
	1	2	3	4
Nonlinearity	104			
Min. algebraic degree of Boolean functions	7			
Max. value of difference distribution table	8			
Max. value of linear approximation table	24			
Algebraic immunity	3			
Number of cycles	4	4	6	4
Minimal cycle length	6	8	4	4

The best known byte permutations with algebraic immunity is equivalent to 3.

Kalyna: linear transformation

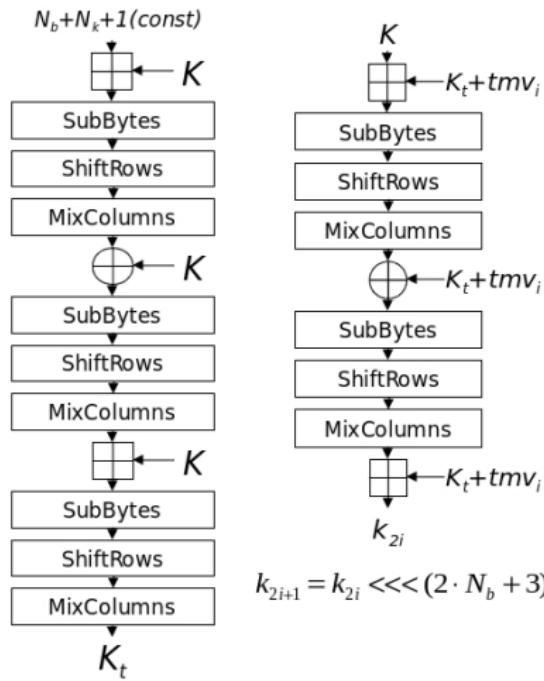
$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 01 \cdot a_0 \oplus 01 \cdot a_1 \oplus 05 \cdot a_2 \oplus 01 \cdot a_3 \oplus 08 \cdot a_4 \oplus 06 \cdot a_5 \oplus 07 \cdot a_6 \oplus 04 \cdot a_7 \\ 04 \cdot a_0 \oplus 01 \cdot a_1 \oplus 01 \cdot a_2 \oplus 05 \cdot a_3 \oplus 01 \cdot a_4 \oplus 08 \cdot a_5 \oplus 06 \cdot a_6 \oplus 07 \cdot a_7 \\ 07 \cdot a_0 \oplus 04 \cdot a_1 \oplus 01 \cdot a_2 \oplus 01 \cdot a_3 \oplus 05 \cdot a_4 \oplus 01 \cdot a_5 \oplus 08 \cdot a_6 \oplus 06 \cdot a_7 \\ 06 \cdot a_0 \oplus 07 \cdot a_1 \oplus 04 \cdot a_2 \oplus 01 \cdot a_3 \oplus 01 \cdot a_4 \oplus 05 \cdot a_5 \oplus 01 \cdot a_6 \oplus 08 \cdot a_7 \\ 08 \cdot a_0 \oplus 06 \cdot a_1 \oplus 07 \cdot a_2 \oplus 04 \cdot a_3 \oplus 01 \cdot a_4 \oplus 01 \cdot a_5 \oplus 05 \cdot a_6 \oplus 01 \cdot a_7 \\ 01 \cdot a_0 \oplus 08 \cdot a_1 \oplus 06 \cdot a_2 \oplus 07 \cdot a_3 \oplus 04 \cdot a_4 \oplus 01 \cdot a_5 \oplus 01 \cdot a_6 \oplus 05 \cdot a_7 \\ 05 \cdot a_0 \oplus 01 \cdot a_1 \oplus 08 \cdot a_2 \oplus 06 \cdot a_3 \oplus 07 \cdot a_4 \oplus 04 \cdot a_5 \oplus 01 \cdot a_6 \oplus 01 \cdot a_7 \\ 01 \cdot a_0 \oplus 05 \cdot a_1 \oplus 01 \cdot a_2 \oplus 08 \cdot a_3 \oplus 06 \cdot a_4 \oplus 07 \cdot a_5 \oplus 04 \cdot a_6 \oplus 01 \cdot a_7 \end{bmatrix}$$

- the branch number is 9 (the MDS matrix)
- effective software and software-hardware implementations

Requirements to Kalyna's Key Schedule

- non-linear dependence of each round key bit on each encryption key bit
- protection from cryptanalytic attacks aimed to key schedule
- high computation complexity of obtaining encryption key having one or several round keys (one-way transformation, additional protection from side-channel attacks)
- key agility is less than three
- possibility to generate round keys in direct and reverse order
- implementation simplicity (the use of transformations from the round function)

Kalyna: Key Schedule



$$tmv_0 = 0x01000100..0100$$
$$tmv_{i+2} = tmv_i \ll 1$$

$$k_{2i+1} = k_{2i} \ll\ll (2 \cdot N_b + 3)$$

$$\Theta^{(K)} = \psi_I \circ \tau_I \circ \pi'_I \circ \eta_I^{(K_\alpha)} \circ \psi_I \circ \tau_I \circ \pi'_I \circ \kappa_I^{(K_\omega)} \circ \psi_I \circ \tau_I \circ \pi'_I \circ \eta_I^{(K_\alpha)}$$

$$\Xi^{(K, K_\sigma, i)} = \eta_I^{(\varphi_i(K_\sigma))} \circ \psi_I \circ \tau_I \circ \pi'_I \circ \kappa_I^{(\varphi_i(K_\sigma))} \circ \psi_I \circ \tau_I \circ \pi'_I \circ \eta_I^{(\varphi_i(K_\sigma))}$$

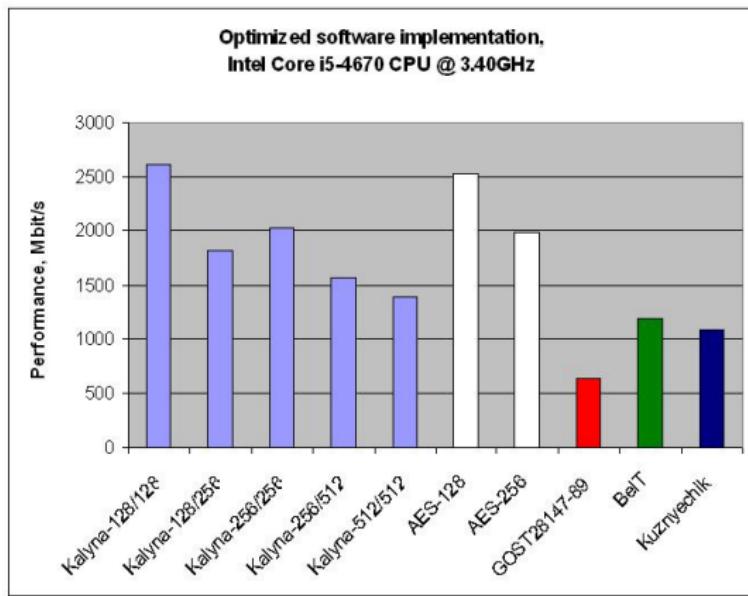
Cryptanalytic attack against Kalyna

Kalyna is resistant to known cryptanalytic methods (based on public information):

- Kalyna-128/128: from 6th round (out of 10)
- Kalyna-128/256: from 10th round* (out of 14)
- Kalyna-256/256: from 7th round (out of 14)
- Kalyna-256/512: from 10th round* (out of 18)
- Kalyna-512/512: from 9th round (out of 18)

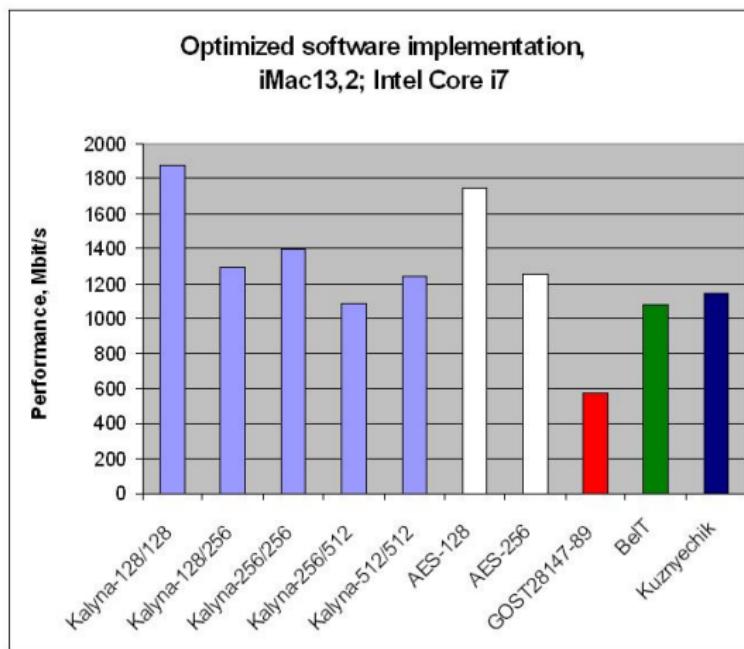
Kalyna: performance comparison with other block ciphers

(Intel Core i5, 64-bit Linux, gcc v4.9.2, best compiler optimization)



<https://github.com/Roman-Oliynykov/ciphers-speed/>

Kalyna: performance comparison with other block ciphers (iMac 13.2, Intel Core i7, best compiler optimization)



<https://github.com/Roman-Oliynykov/ciphers-speed/>

NIST STS of Kalyna's output sequences

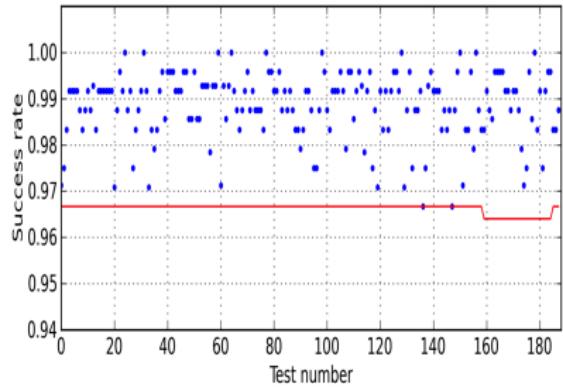


Figure: Even round keys

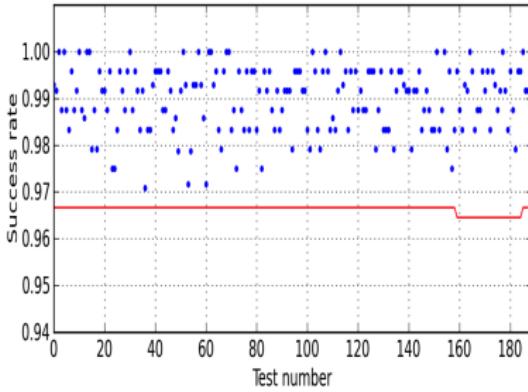


Figure: CTR encryption

DSTU 7624:2014: modes of operation

#	Description	Name	Property
1	Electronic Codebook	ECB	confidentiality
2	Counter	CRT	confidentiality
3	Cipher Feedback	CFB	confidentiality
4	Cipher-based Message Authentication Code	CMAC	integrity
5	Cipher Block Chaining	CBC	confidentiality
6	Output Feedback	OFB	confidentiality
7	Galois Counter Mode	GCM	confidentiality and integrity
7	Galois Message Authentication Code	GMAC	integrity
8	Counter with CBC-MAC	CCM	confidentiality and integrity
9	XEX-based tweaked-codebook mode with ciphertext stealing	XTS	confidentiality
10	Key Wrap	KW	confidentiality and integrity

- Ten modes of operation for the new block cipher
 - ISO 10116: ECB, CBC, CFB, OFB, CTR
 - additional modes, simplified/improved comparing to NIST SP 800-38: GCM/GMAC, CCM, XTS, KW
- Test vectors (including not aligned to the block length and, for some modes, byte length)
- Requirements to implementation:
 - general concepts paying developer's attention to take steps for prevention of side-channel attacks, timing attacks, CRIME/BREACH specific vulnerabilities, etc.
 - limits on the total number of invocation of the block cipher during the encryption key lifetime
 - message replay prevention
- etc.

Conclusions

The Kalyna block cipher provides

- normal, high and ultra high security level
- transparent construction and conservative design
- fast and effective software and software-hardware implementations on modern 64-bit platforms
- optimized construction for better performance on encryption and decryption for CTR, CFB, CMAC, OFB, GCM, GMAC, CCM
- a new construction of key schedule based on the round transformation
- common look-up tables with the hash function "Kupyna" (the new Ukrainian standard DSTU 7564:2014)

Besides the block cipher, the new Ukrainian standard DSTU 7624:2014 defines ten modes of operation, test vectors, requirements for implementation, limits on protected information amount for a single key application, etc.

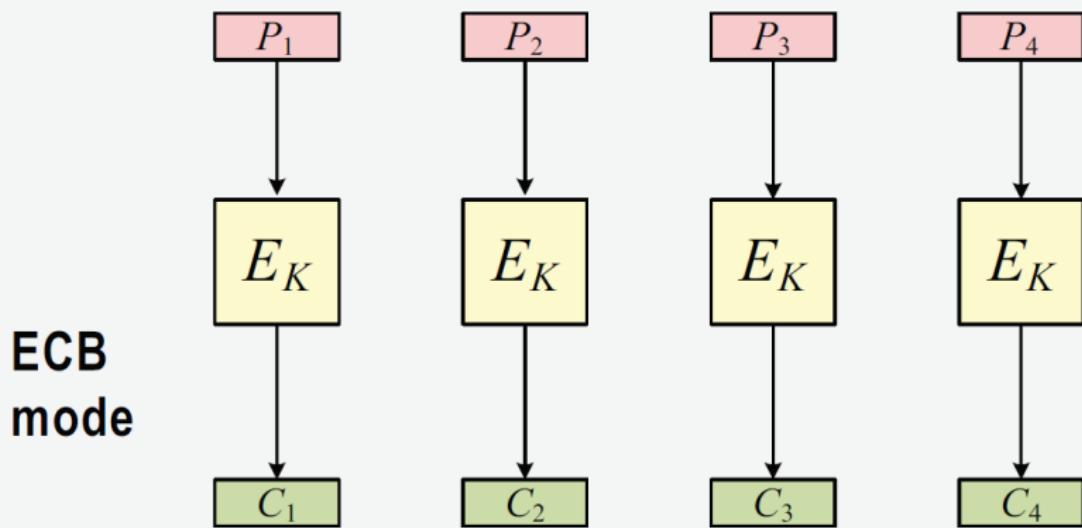


Figure: Kalyna / Viburnum / Krossvedslekta

Stream modes of block ciphers

- Detailed description at [NIST](#)
- Evaluation [results](#) by Phillip Rogaway

Electronic Code Book (ECB)



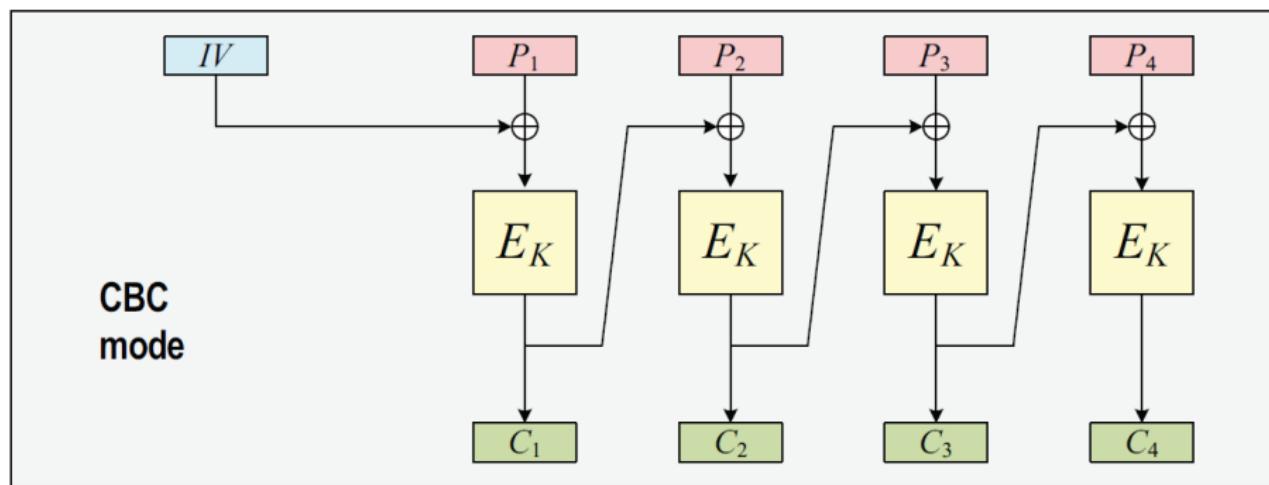
```

10 algorithm ECBK(P)
11 //  $K \in \mathcal{K}$ ,  $P \in (\{0, 1\}^n)^+$ 
12  $P_1 \dots P_m \leftarrow P$  where  $|P_i| = n$ 
13 for  $i \leftarrow 1$  to  $m$  do  $C_i \leftarrow E_K(P_i)$ 
14  $C \leftarrow C_1 \dots C_m$ 
15 return  $C$ 

```

ECB mode

Cipher Block Chaining (CBC)



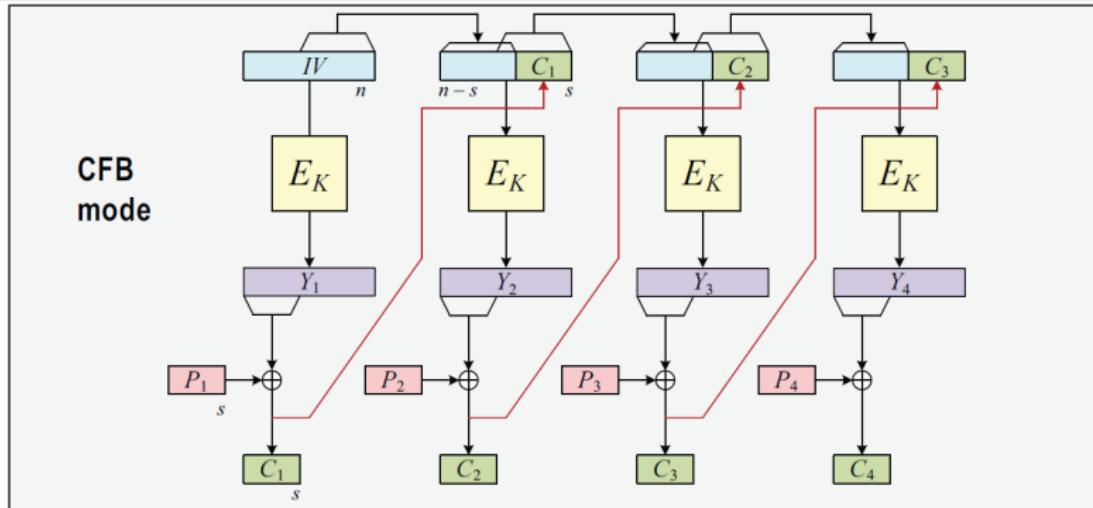
```

10 algorithm CBCIVK(P)
11 //  $K \in \mathcal{K}$ ,  $IV \in \{0,1\}^n$ ,  $P \in (\{0,1\}^n)^+$ 
12  $P_1 \dots P_m \leftarrow P$  where  $|P_i| = n$ 
13  $C_0 \leftarrow IV$ 
14 for  $i \leftarrow 1$  to  $m$  do  $C_i \leftarrow E_K(P_i \oplus C_{i-1})$ 
15  $C \leftarrow C_1 \dots C_m$ 
16 return  $C$ 

```

CBC mode

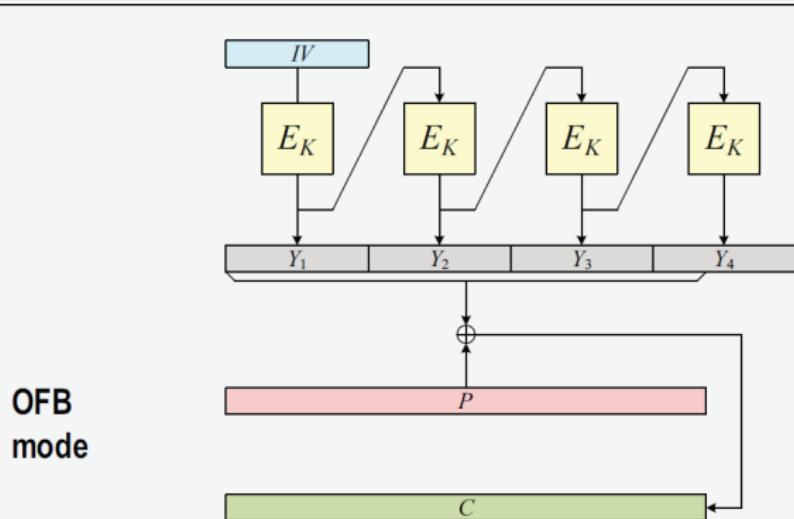
Cipher Feedback (CFB)



```

20 algorithm CFBIVK(P) CFB mode
21 //  $K \in \mathcal{K}$ ,  $IV \in \{0,1\}^n$ ,  $P \in (\{0,1\}^s)^+$ 
22  $P_1 \dots P_m \leftarrow P$  where  $|P_i| = s$ 
23  $X_1 \leftarrow IV$ 
24 for  $i \leftarrow 1$  to  $m$  do
25    $Y_i \leftarrow E_K(X_i)$ 
26    $C_i \leftarrow P_i \oplus \text{MSB}_s(Y_i)$ 
27    $X_{i+1} \leftarrow \text{LSB}_{n-s}(X_i) \parallel C_i$ 
28  $C \leftarrow C_1 \dots C_m$ 
29 return C
  
```

Output Feedback (OFB)



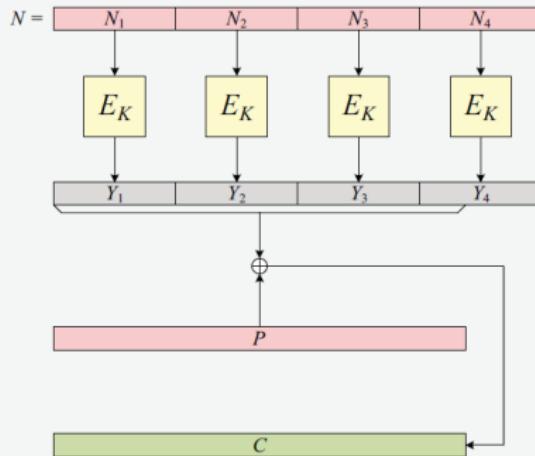
```

30 algorithm  $\text{OFB}_K^{IV}(P)$ 
31 //  $K \in \mathcal{K}$ ,  $IV \in \{0, 1\}^n$ ,  $P \in \{0, 1\}^*$ 
32  $m \leftarrow \lceil |P|/n \rceil$ 
33  $Y_0 \leftarrow IV$ 
34 for  $i \leftarrow 1$  to  $m$  do  $Y_i \leftarrow E_K(Y_{i-1})$ 
35  $Y \leftarrow \text{MSB}_{|P|}(Y_1 \dots Y_m)$ 
36  $C \leftarrow P \oplus Y$ 
37 return  $C$ 

```

OFB mode

Counter (CTR)



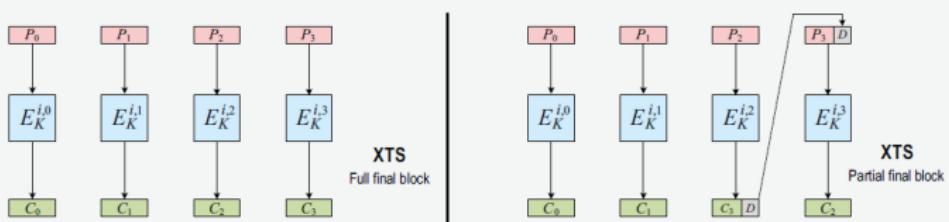
```

10 algorithm  $\text{CTR}_K^N(P)$ 
11 //  $K \in \mathcal{K}$ ,  $N \in \{0, 1\}^n$ ,  $P \in \{0, 1\}^*$ ,  $|N| = \lceil |P|/n \rceil$ 
12  $m \leftarrow \lceil |P|/n \rceil$ 
13  $(N_1, \dots, N_m) \leftarrow N$ 
14 for  $i \leftarrow 1$  to  $m$  do  $Y_i \leftarrow E_K(N_i)$ 
15  $Y \leftarrow \text{MSB}_{|P|}(Y_1 \dots Y_m)$ 
16  $C \leftarrow P \oplus Y$ 
17 return  $C$ 

```

CTR mode

XTS



```

10 algorithm XTSiK(P) XTS mode
11 //  $K \in \mathcal{K}$ ,  $i \in \{0, 1\}^{128}$ ,  $P \in \{0, 1\}^N$ 
12  $P_0 P_1 \dots P_m \leftarrow P$  where  $m = \lceil |P|/n \rceil - 1$  and  $|P_j| = n$  for all  $0 \leq j < m$ ,  $1 \leq |P_m| \leq n$ 
13  $b \leftarrow |P_m|$ 
14 for  $j \leftarrow 0$  to  $m - 1$  do  $C_j \leftarrow \text{XEX2}_K^{i,j}(P_j)$ 
15 if  $b = n$  then Full final block
16    $C_m \leftarrow \text{XEX2}_K^{i,m}(P_m)$ 
17 else Partial final block
18    $C_m \parallel D \leftarrow C_{m-1}$  where  $|C_m| = b$ 
19    $C_{m-1} \leftarrow \text{XEX2}_K^{i,m}(P_m \parallel D)$ 
20 endif
21 return  $C_0 \dots C_m$ 

30 algorithm XEX2i,jK(X) Two-key version of XEX
31  $K1 \parallel K2 \leftarrow K$  where  $|K1| = |K2|$ 
32  $L \leftarrow E_{K2}(i)$ 
33  $\Delta \leftarrow L \cdot \alpha^j$ 
34 return  $E_{K1}(X \oplus \Delta) \oplus \Delta$ 

```

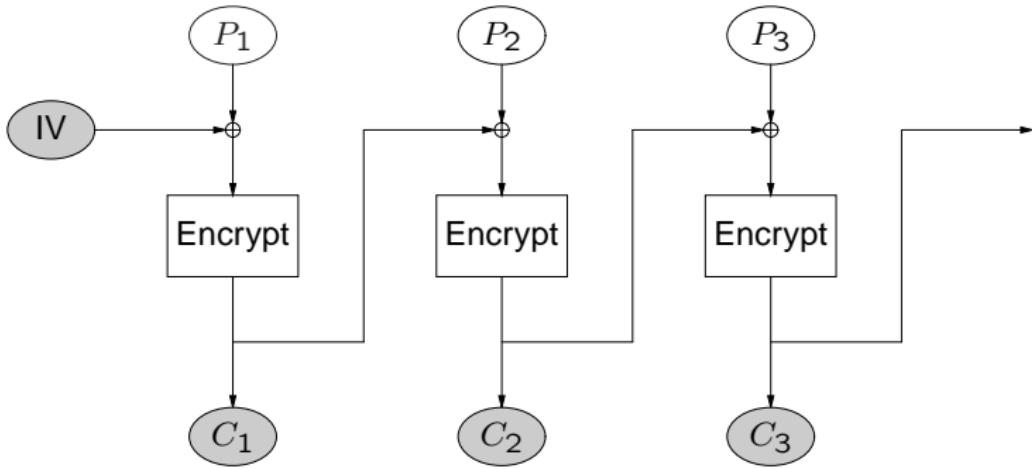
Modes of Operation

- Direct use of a block cipher is inadvisable
- Enemy can build up “code book” of plaintext/ciphertext equivalents
- Beyond that, direct use only works on messages that are a multiple of the cipher block size in length
- Solution: five standard *Modes of Operation*: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR).

Electronic Code Book

- Direct use of the block cipher
- Used primarily to transmit encrypted keys
- Very weak if used for general-purpose encryption; never use it for a file or a message.
- Attacker can build up codebook; no semantic security
- We write $\{P\}_k \rightarrow C$ to denote “encryption of plaintext P with key k to produce ciphertext C ”

Cipher Block Chaining



$$\begin{aligned} \{P_i \oplus C_{i-1}\}_k &\rightarrow C_i \\ \{C_i\}_{k-1} \oplus C_{i-1} &\rightarrow P_i \end{aligned}$$

Properties of CBC

- The ciphertext of each encrypted block depends on the IV and the plaintext of all preceding blocks.
- There is a dummy initial ciphertext block C_0 known as the *Initialization Vector* (IV); the receiver must know this value.
- Consider a 4-block message:

$$\begin{aligned}C_1 &= \{P_1 \oplus IV\}_k \\C_2 &= \{P_2 \oplus C_1\}_k \\C_3 &= \{P_3 \oplus C_2\}_k \\C_4 &= \{P_4 \oplus C_3\}_k\end{aligned}$$

If C_2 is damaged during transmission, what happens to the plaintext?

Error Propagation in CBC Mode

- Look at the decryption process, where C' is a corrupted version of C :

$$\begin{aligned}P_1 &= \{C_1\}_{k-1} \oplus IV \\P_2 &= \{\textcolor{red}{C'_2}\}_{k-1} \oplus C_1 \\P_3 &= \{C_3\}_{k-1} \oplus \textcolor{red}{C'_2} \\P_4 &= \{C_4\}_{k-1} \oplus C_3\end{aligned}$$

- P_1 depends only on C_1 and IV , and is unaffected
- P_2 depends on C_2 and C_1 , and hence is corrupted
- P_3 depends on C_3 and C_2 , and is also corrupted. The enemy can control the change to P_3 .
- P_4 depends on C_4 and C_3 , and not C_2 ; it thus isn't affected.
- Conclusion: Two blocks change, one of them predictably

Cutting and Pasting CBC Messages

- Consider the encrypted message

$$IV, C_1, C_2, C_3, C_4, C_5$$

- The shortened message IV, C_1, C_2, C_3, C_4 appears valid
- The truncated message C_2, C_3, C_4, C_5 is valid: C_2 acts as the IV.
- Even C_2, C_3, C_4 is valid, and will decrypt properly.
- Any subset of a CBC message will decrypt cleanly.
- If we snip out blocks, leaving IV, C_1, C_4, C_5 , we only corrupt one block of plaintext.
- Conclusion: if you want message integrity, you have to do it yourself.

The IV

- The IV provides semantic security: identical messages have different ciphertexts
- Common message start is also hidden
- The IV *must* be unpredictable by the enemy
- Good strategy: have a random key lying around; encrypt a counter to produce the IV
- Note: since the IV is transmitted unencrypted, the other side need not know this key

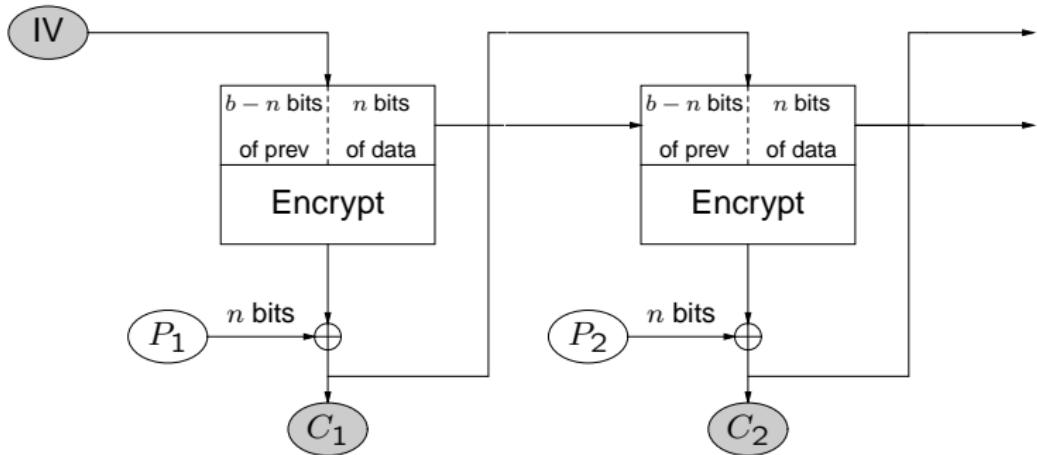
CBC Padding

- CBC mode requires the input to be a multiple of the cipher's block size.
- Must (somehow) handle other lengths
- Usual strategy: (securely) transmit explicit input length
- Option: *Cipher-Text Stealing* (see RFC 2040). Does not increase message size

CBC MAC

- Recall that for CBC encryption, the last block of ciphertext depends on all of the plaintext
- Do a second encryption (using a different key), but only send the last block
- If you use the same key, a CBC cut-and-paste attack will work
- Query: what IV should you use? (It doesn't matter; a constant IV will suffice.)
- Warning: if message sizes can vary, *prepend* the length
- NIST standard CMAC uses slightly more complex scheme that avoids needing to know the length in advance

n-bit Cipher Feedback

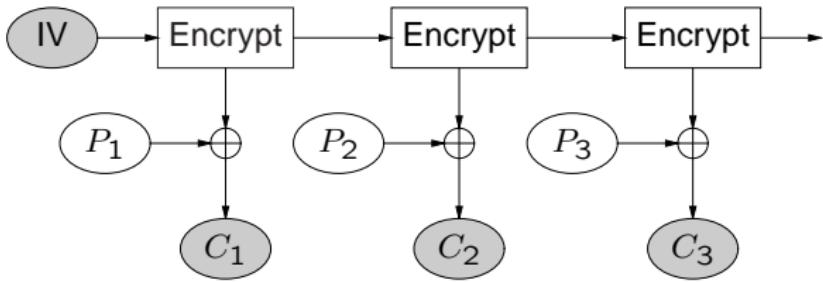


$$\begin{aligned} P_i \oplus \{C_{i-1}\}_k &\rightarrow C_i \\ \{C_{i-1}\}_k \oplus C_i &\rightarrow P_i \end{aligned}$$

Properties of Cipher Feedback Mode

- Underlying block cipher used only in encryption mode
- Feedback path actually incorporates a shift register: shift old cipher input left by n bits; insert first n bits of previous ciphertext output
- 8-bit CFB is good for asynchronous terminal traffic — but requires one encryption for each *byte* of plaintext
- Errors propagate while bad data is in the shift register — 17 bytes for CFB₈ when using AES.
- Copes gracefully with deletion of n -bit unit
- Interesting uses: CFB₁, CFB₈, CFB₁₂₈
- IV selected the same way as in CBC mode

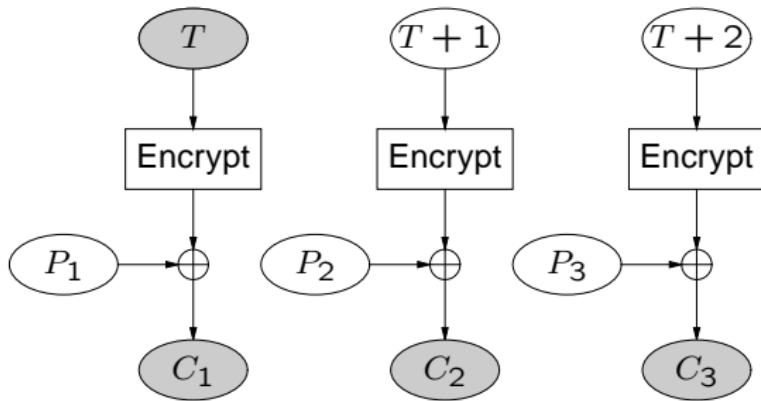
n-bit Output Feedback



Properties of Output Feedback Mode

- No error propagation
- Active attacker can make controlled changes to plaintext
- OFB is a form of *stream cipher*

Counter Mode



Properties of Counter Mode

- Another form of stream cipher
- Frequently split the counter into two sections: message number and block number within the message
- Active attacker can make controlled changes to plaintext
- Highly parallelizable; no linkage between stages
- Vital that counter never repeat for any given key

Which Mode for What Task?

- General file or packet encryption: CBC.
☞ Input must be padded to multiple of cipher block size
- Risk of byte or bit deletion: CFB₈ or CFB₁
- Bit stream; noisy line and error propagation is undesirable: OFB
- Very high-speed data: CTR
- In most situations, an integrity check is needed

Integrity Checks

- Actually, integrity checks are almost always needed
- Frequently, attacks on integrity can be used to attack confidentiality
- Usual solution: use separate integrity check along with encryption
- Simple solutions don't work
- Choices: HMAC, CBC MAC, CMAC, combined modes of operation, lesser-known schemes
- One bad idea: append a cryptographic hash to some plaintext, and encrypt the whole thing with, say, CBC mode

$$\{P \parallel H(P)\}_K$$

- This can fall victim to a *chosen plaintext attack*

Chosen Plaintext Attack

- The enemy picks some plaintext P and tricks you into encrypting it
- This has happened in the real world!
- You transmit

$$\{\text{prefix} \parallel P \parallel \text{suffix} \parallel H(\text{prefix} \parallel P \parallel \text{suffix})\}_K$$

- But P is of the form

$$\text{prefix} \parallel P' \parallel \text{suffix} \parallel H(\text{prefix} \parallel P' \parallel \text{suffix})$$

- An ordinary CBC subset will have the checksum!

CCM Mode

- Inputs: nonce N , additional data A (e.g., network packet headers), plaintext P
- Authenticates A and P ; encrypts P
- Calculate $T \leftarrow \text{CBC-MAC}_K(N \parallel A \parallel P)$
- Ciphertext is $\text{CTR}_K(P)$; MAC is $\text{CTR}_K(T)$
- Note: the first counter value is used to encrypt T
- Note: N acts as an IV; it must be non-repeating but need not be secret or unpredictable

Stream Ciphers

- Key stream generator produces a sequence S of pseudo-random bytes; key stream bytes are combined (generally via XOR) with plaintext bytes
- $P_i \oplus S_i \rightarrow C_i$
- Stream ciphers are very good for asynchronous traffic
- Best-known stream cipher is RC4; commonly used with SSL
- Key stream S must *never* be reused for different plaintexts:

$$\begin{aligned}C &= A \oplus K \\C' &= B \oplus K \\C \oplus C' &= A \oplus K \oplus B \oplus K \\&= A \oplus B\end{aligned}$$

- Guess at A and see if B makes sense; repeat for subsequent bytes

RC4

- Extremely efficient
- After key setup, it just produces a key stream
- No way to resynchronize except by rekeying and starting over
- Internal state is a 256-byte array plus two integers
- Note: weaknesses if used in ways other than as a stream cipher.

The Guts of RC4

```
for(counter = 0; counter < buffer_len; counter ++)
{
    x = (x + 1) % 256;
    y = (state[x] + y) % 256;
    swap_byte(&state[x], &state[y]);
    xorIndex = (state[x] + state[y]) % 256;
    buffer_ptr[counter] ^= state[xorIndex];
}
```

Hash functions and MAC

Hash function

Definition

A cryptographic hash function h is a function which takes arbitrary length bit strings as input and produces a fixed length bit string as output, called a hashcode or hash value, such that

- ① h is preimage resistant, i.e. it should be hard to find a message with a given hash value,
- ② h is collision resistant, i.e. it should be hard to find two messages with the same hash value,
- ③ h is second preimage resistant, i.e. given one message it should be hard to find another message with the same hash value.

Merkle–Damgård Construction

Let f be a compression function from s bits to n bits, with $s > n$, which is believed to be collision resistant. We wish to use f to construct a hash function h which takes arbitrary length inputs, and which produces hash codes of n bits in length.

$$l = s - n$$

Pad the input message m with zeros so that it is a multiple of l bits in length

Divide the input m into t blocks of l bits long, m_1, \dots, m_t

Set H to be some fixed bit string of length n .

```
for i = 1 to t do
    | H = f(H || mi)
end
return (H)
```

Examples of hash functions

- MD4: This has 3 rounds of 16 steps and an output bitlength of 128 bits.
- MD5: This has 4 rounds of 16 steps and an output bitlength of 128 bits.
- SHA-1: This has 4 rounds of 20 steps and an output bitlength of 160 bits.
- RIPEMD-160: This has 5 rounds of 16 steps and an output bitlength of 160 bits.
- SHA-256: This has 64 rounds of single steps and an output bitlength of 256 bits.
- SHA-384: This is identical to SHA-512 except the output is truncated to 384 bits.
- SHA-512: This has 80 rounds of single steps and an output bitlength of 512 bits.
- SHA-3

SHA-3

- Selected on October 02, 2012 by the National Institute of Standards and Technology (NIST) as a new cryptographic hash algorithm, one of the fundamental tools of modern information security.
- The winning algorithm, *Keccak* (pronounced “catch-ack”), was created by Guido Bertoni, Joan Daemen and Gilles Van Assche of STMicroelectronics and Michal Peeters of NXP Semiconductors.
- The team's entry beat out 63 other submissions that NIST received after its open call for candidate algorithms in 2007, when it was thought that SHA-2, the standard secure hash algorithm, might be threatened.
- See [NIST FIPS 202](#) for detailed description.

Outline

- 1 The SHA-3 competition
- 2 The sponge construction
- 3 Inside KECCAK
- 4 The SHA-3 FIPS

Outline

1 The SHA-3 competition

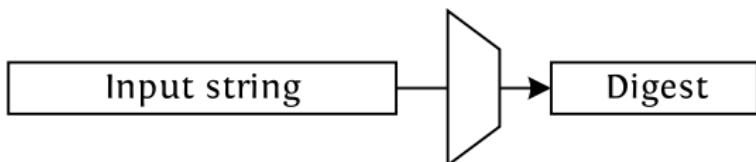
2 The sponge construction

3 Inside KECCAK

4 The SHA-3 FIPS

Cryptographic hash functions

- Function h from \mathbb{Z}_2^* to \mathbb{Z}_2^n
- Typical values for n : 128, 160, 256, 512



- Pre-image resistant: it shall take 2^n effort to
 - given y , find x such that $h(x) = y$
- 2nd pre-image resistance: it shall take 2^n effort to
 - given M and $h(M)$, find another M' with $h(M') = h(M)$
- Collision resistance: it shall take $2^{n/2}$ effort to
 - find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$
- More general: *should behave like a random oracle*

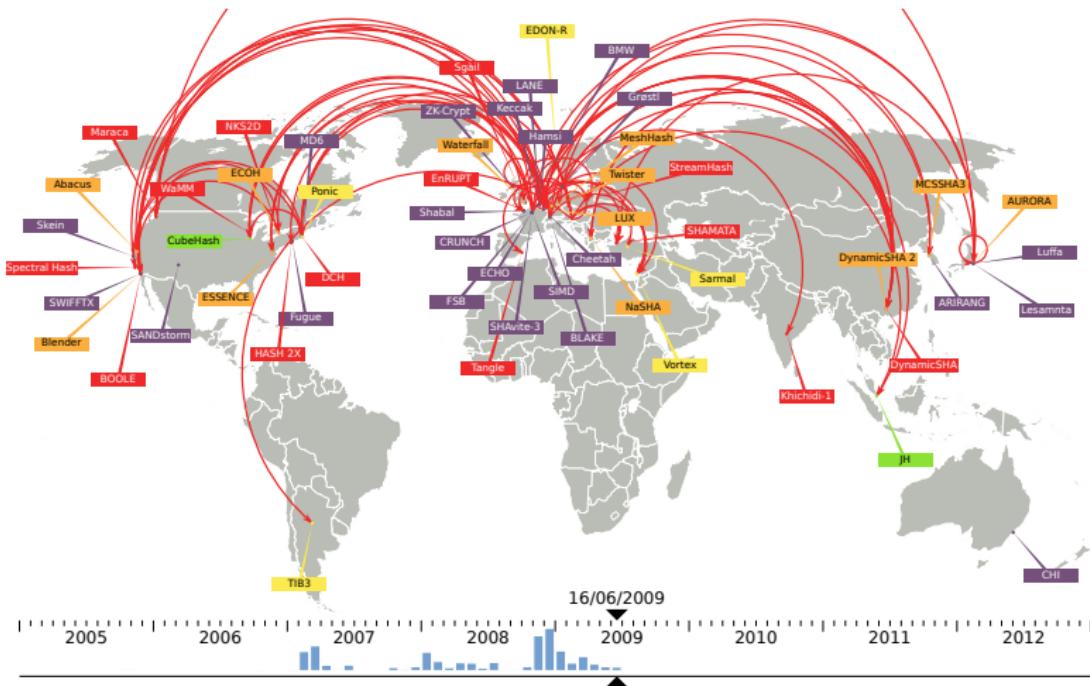
The origins of the SHA-3 competition

- 2005-2006: NIST thinks about having a SHA-3 contest
 - MD5 and standard SHA-1 were damaged by attacks
 - SHA-2 based on the same principles than MD5 and SHA-1
 - open call for SHA-2 successor
 - ...and for analysis, comparisons, etc.
- October 2008: Deadline for proposals
 - *more efficient than SHA-2*
 - output lengths: 224, 256, 384, 512 bits
 - security: collision and (2nd) pre-image resistant
 - specs, reference and optimized code, test vectors
 - design rationale and preliminary analysis
 - patent waiver

The SHA-3 competition

- First round: October 2008 to summer 2009
 - 64 submissions, 51 accepted
 - 37 presented at 1st SHA-3 candidate conf. in Leuven, February 2009
 - many broken by cryptanalysis
 - NIST narrowed down to 14 semi-finalists
- Second round: summer 2009 to autumn 2010
 - analysis presented at 2nd SHA-3 conf. in Santa Barbara, August 2010
 - NIST narrowed down to 5 finalists
- Third round: autumn 2010 to October 2012
 - analysis presented at 3rd SHA-3 conf. in Washington, March 2012
- October 2, 2012: NIST announces KECCAK as SHA-3 winner

NIST SHA-3: the battlefield



[courtesy of Christophe De Cannière]

Outline

1 The SHA-3 competition

2 The sponge construction

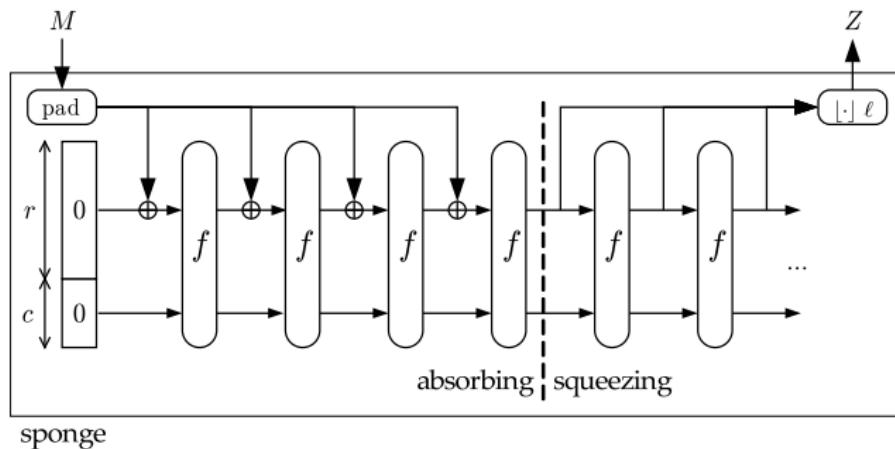
3 Inside KECCAK

4 The SHA-3 FIPS

Sponge origin: RADIOGATÚN

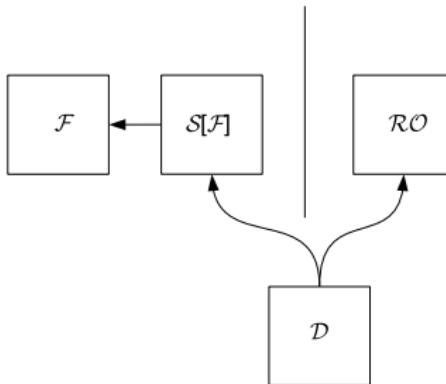
- Initiative to design hash/stream function (late 2005)
 - rumours about NIST call for hash functions
 - starting point: **fixing PANAMA** [Daemen, Clapp, FSE 1998]
 - with long-time colleagues **Gilles Van Assche** and **Michaël Peeters**
 - and ST Italy colleague **Guido Bertoni** joining in
- **RADIOGATÚN** [Keccak team, NIST 2nd hash workshop 2006]
 - more conservative than PANAMA
 - **arbitrary output length**
 - **expressing security claim** for arbitrary output length function
- Sponge functions [Keccak team, Ecrypt hash, 2007]
 - **random sponge** instead of random oracle as security goal
 - sponge construction calling random permutation
 - ... *closest thing to a random oracle with a finite state ...*

The sponge construction



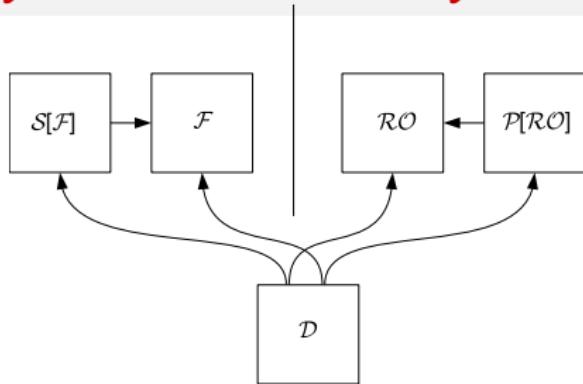
- Generalizes hash function: *extendable output function* (XOF)
- Calls a b -bit **permutation** f , with $b = r + c$
 - r bits of *rate*
 - c bits of *capacity* (security parameter)

Generic security: indistinguishability



- Success probability of distinguishing between:
 - ideal function: a monolithic random oracle \mathcal{RO}
 - construction $\mathcal{S}[\mathcal{F}]$ calling an random permutation \mathcal{F}
- Adversary \mathcal{D} sends queries (M, ℓ) according to algorithm
- Express $\Pr(\text{success} | \mathcal{D})$ as a function of total cost of queries N
- **Problem:** in real world, \mathcal{F} is available to adversary

Generic security: indifferentiability [Maurer et al. (2004)]



- Applied to hash functions in [Coron et al. (2005)]
 - distinguishing mode-of-use from ideal function (\mathcal{RO})
 - covers adversary with access to permutation \mathcal{F} at left
 - additional interface, covered by a *simulator* at right
- Methodology:
 - build \mathcal{P} that makes left/right distinguishing difficult
 - prove bound for advantage given this simulator \mathcal{P}
 - \mathcal{P} may query \mathcal{RO} for acting \mathcal{S} -consistently: $\mathcal{P}[\mathcal{RO}]$

Generic security of the sponge construction

Concept of *advantage*:

$$\Pr(\text{success}|\mathcal{D}) = \frac{1}{2} + \frac{1}{2}\text{Adv}(\mathcal{D})$$

Theorem (Bound on the \mathcal{RO} -differentiating advantage of sponge)

$$A \leq \frac{N^2}{2^{c+1}}$$

A: differentiating advantage of random sponge from random oracle

N: total data complexity

c: capacity

[Keccak team, Eurocrypt 2008]

Implications of the bound

- Let \mathcal{D} : n -bit output pre-image attack. Success probability:
 - for random oracle: $P_{\text{pre}}(\mathcal{D}|\mathcal{RO}) = q2^{-n}$
 - for random sponge: $P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) = ?$
- A distinguisher \mathcal{D} with $A = P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) - P_{\text{pre}}(\mathcal{D}|\mathcal{RO})$
 - do pre-image attack
 - if success, conclude random sponge and \mathcal{RO} otherwise
- But we have a proven bound $A \leq \frac{N^2}{2^{c+1}}$, so

$$P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) \leq P_{\text{pre}}(\mathcal{D}|\mathcal{RO}) + \frac{N^2}{2^{c+1}}$$

- Can be generalized to any attack
- Note that A is independent of output length n

Implications of the bound (cont'd)

- Informally, random sponge is like random oracle for $N < 2^{c/2}$
- Security strength for output length n :
 - collision-resistance: $\min(c/2, n/2)$
 - first pre-image resistance: $\min(c/2, n)$
 - second pre-image resistance: $\min(c/2, n)$
- Proof assumes f is a **random** permutation
 - provably secure against generic attacks
 - ...but not against attacks that exploit specific properties of f
- No security against multi-stage adversaries

Design approach

Hermetic sponge strategy

- Instantiate a **sponge function**
- Claim a security level of $2^{c/2}$

Our mission

Design permutation f without exploitable properties

How to build a strong permutation

- Like a block cipher
 - Sequence of identical rounds
 - Round consists of sequence of simple step mappings
- ...but not quite
 - No key schedule
 - Round constants instead of round keys
 - Inverse permutation need not be efficient

Outline

1 The SHA-3 competition

2 The sponge construction

3 Inside KECCAK

4 The SHA-3 FIPS

KECCAK[r, c]

- Sponge function using the **permutation** KECCAK- f
 - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
... from toy over lightweight to high-speed ...
- SHA-3 instance: $r = 1088$ and $c = 512$
 - permutation width: 1600
 - security strength 256: post-quantum sufficient
- Lightweight instance: $r = 40$ and $c = 160$
 - permutation width: 200
 - security strength 80: same as (initially expected from) SHA-1

See [The KECCAK reference] for more details

KECCAK[r, c]

- Sponge function using the **permutation** KECCAK- f
 - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
... from toy over lightweight to high-speed ...
- SHA-3 instance: $r = 1088$ and $c = 512$
 - permutation width: 1600
 - security strength 256: post-quantum sufficient
- Lightweight instance: $r = 40$ and $c = 160$
 - permutation width: 200
 - security strength 80: same as (initially expected from) SHA-1

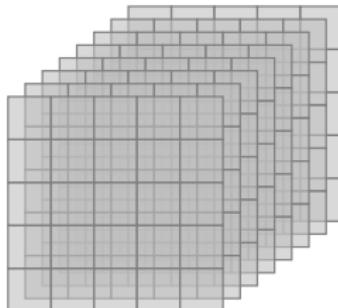
See [The KECCAK reference] for more details

KECCAK[r, c]

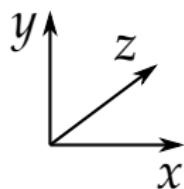
- Sponge function using the **permutation** KECCAK- f
 - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
... from toy over lightweight to high-speed ...
- SHA-3 instance: $r = 1088$ and $c = 512$
 - permutation width: 1600
 - security strength 256: post-quantum sufficient
- Lightweight instance: $r = 40$ and $c = 160$
 - permutation width: 200
 - security strength 80: same as (initially expected from) SHA-1

See [The KECCAK reference] for more details

The state: an array of $5 \times 5 \times 2^\ell$ bits

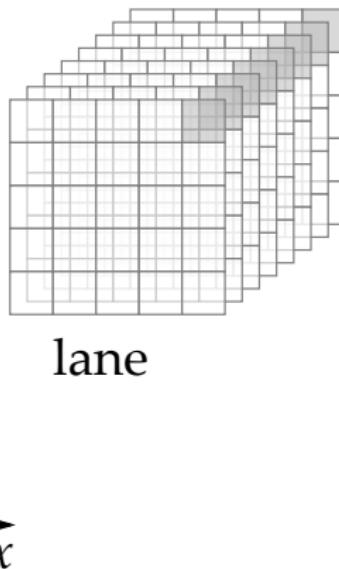


state



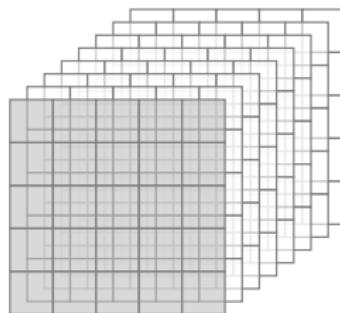
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5) -bit slices, 2^ℓ of them

The state: an array of $5 \times 5 \times 2^\ell$ bits

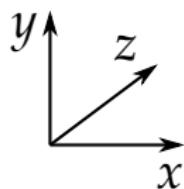


- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5) -bit slices, 2^ℓ of them

The state: an array of $5 \times 5 \times 2^\ell$ bits

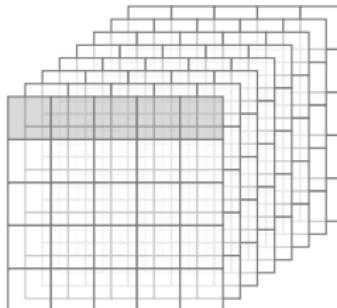


slice

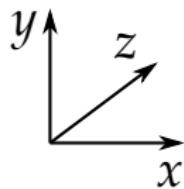


- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5) -bit slices, 2^ℓ of them

The state: an array of $5 \times 5 \times 2^\ell$ bits

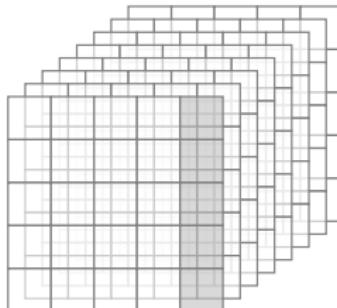


row

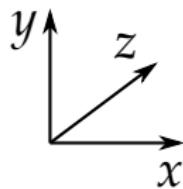


- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5) -bit slices, 2^ℓ of them

The state: an array of $5 \times 5 \times 2^\ell$ bits

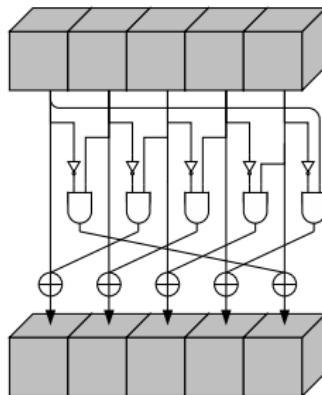


column



- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5) -bit slices, 2^ℓ of them

χ , the nonlinear mapping in KECCAK-f



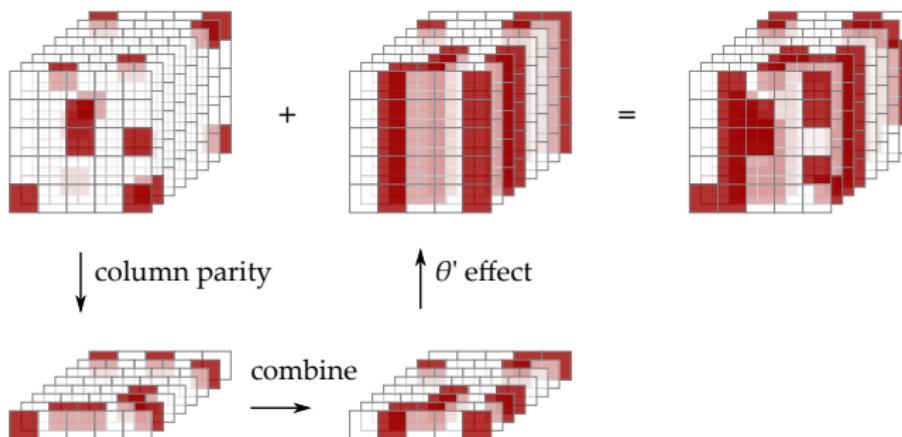
- “Flip bit if neighbors exhibit 01 pattern”
- Operates independently and in parallel on 5-bit rows
- **Cheap:** small number of operations per bit

θ' , a first attempt at mixing bits

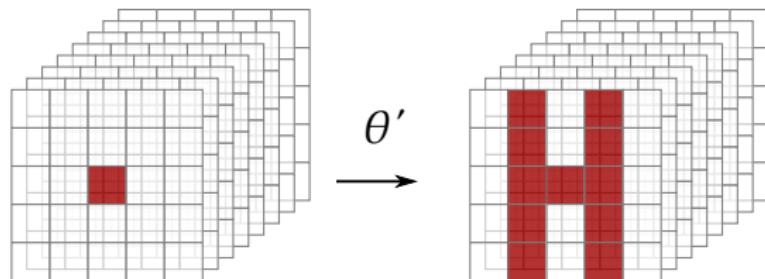
- Compute parity $c_{x,z}$ of each column
- Add to each cell parity of neighboring columns:

$$b_{x,y,z} = a_{x,y,z} \oplus c_{x-1,z} \oplus c_{x+1,z}$$

- **Cheap:** two XORs per bit

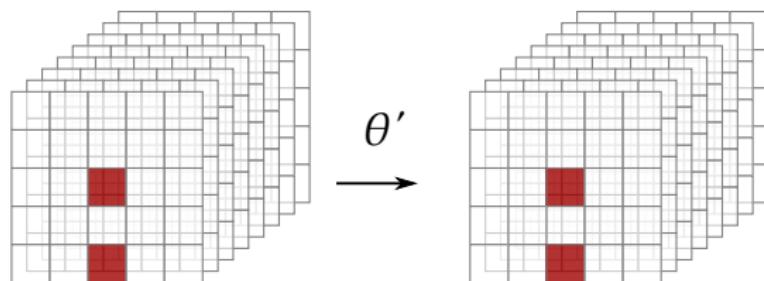


Diffusion of θ'



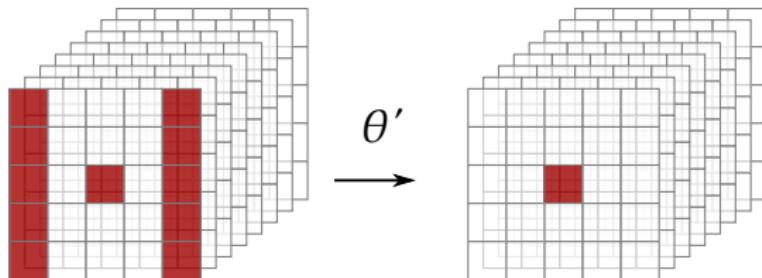
$$\begin{aligned} & 1 + (1 + y + y^2 + y^3 + y^4) (x + x^4) \\ & (\text{mod } \langle 1 + x^5, 1 + y^5, 1 + z^w \rangle) \end{aligned}$$

Diffusion of θ' (kernel)



$$\begin{aligned} & 1 + (1 + y + y^2 + y^3 + y^4) (x + x^4) \\ & (\text{mod } \langle 1 + x^5, 1 + y^5, 1 + z^w \rangle) \end{aligned}$$

Diffusion of the inverse of θ'



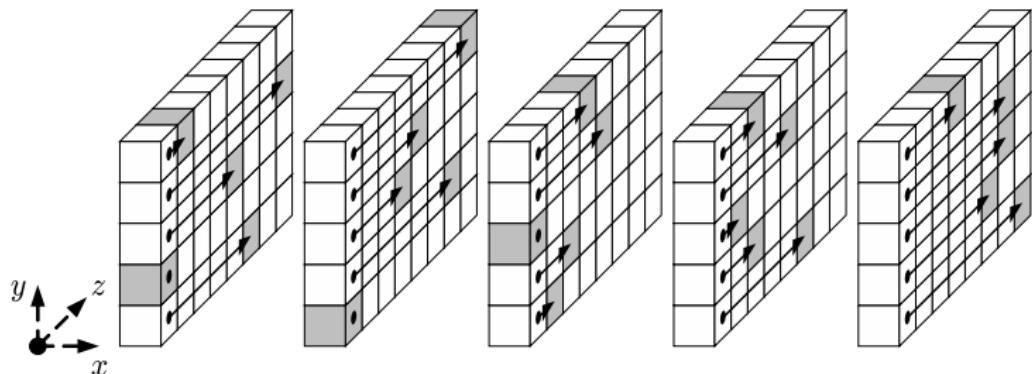
$$1 + (1 + y + y^2 + y^3 + y^4) (x^2 + x^3) \pmod{\langle 1 + x^5, 1 + y^5, 1 + z^w \rangle}$$

ρ for inter-slice dispersion

- We need diffusion between the slices ...
- ρ : cyclic shifts of lanes with offsets

$$i(i+1)/2 \bmod 2^\ell, \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^{i-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- Offsets cycle through all values below 2^ℓ

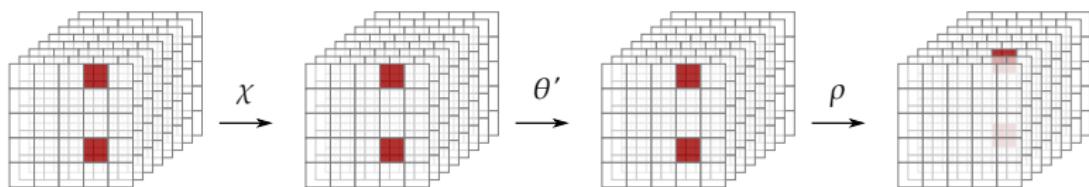


ι to break symmetry

- XOR of round-dependent constant to lane in origin
- Without ι , the round mapping would be symmetric
 - invariant to translation in the z-direction
 - susceptible to *rotational* cryptanalysis
- Without ι , all rounds would be the same
 - susceptibility to *slide* attacks
 - defective cycle structure
- Without ι , we get simple fixed points (000 and 111)

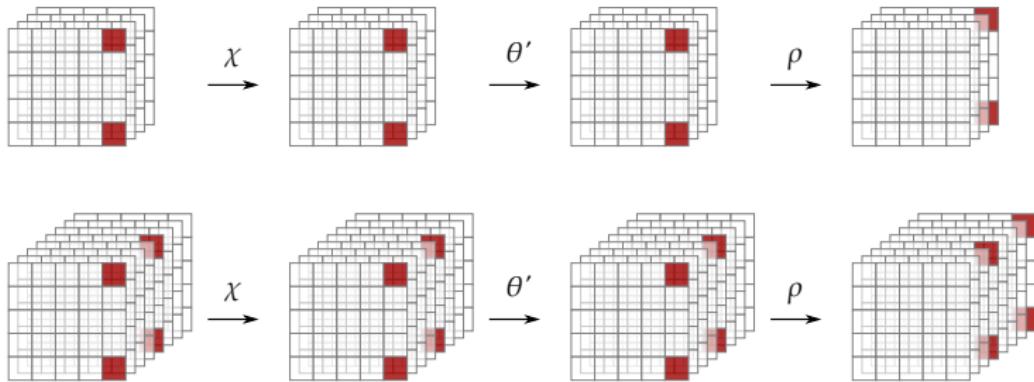
A first attempt at KECCAK-*f*

- Round function: $R = \iota \circ \rho \circ \theta' \circ \chi$
- Problem: low-weight periodic trails by chaining:



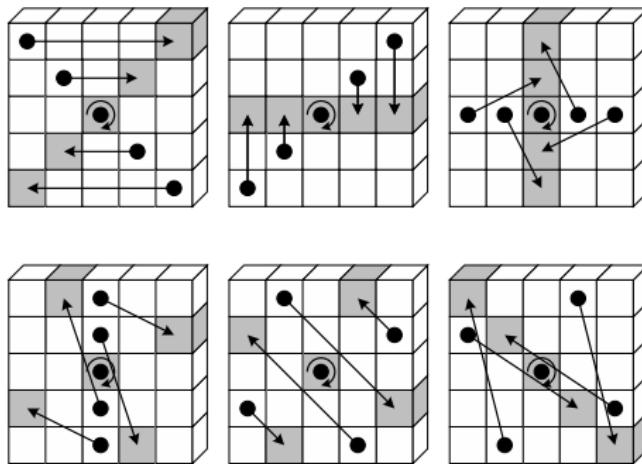
- χ : propagates unchanged with weight 4
- θ' : propagates unchanged, because all column parities are 0
- ρ : in general moves active bits to different slices ...
...but not always

The Matryoshka property



- Patterns in Q' are z-periodic versions of patterns in Q
- Weight of trail Q' is twice that of trail Q (or 2^n times in general)

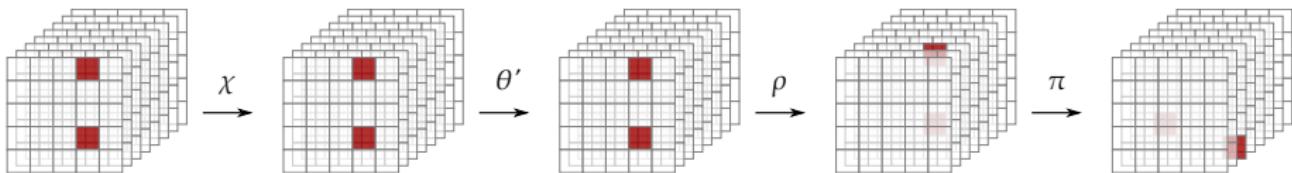
π for disturbing horizontal/vertical alignment



$$a_{x,y} \leftarrow a_{x',y'} \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

A second attempt at KECCAK-*f*

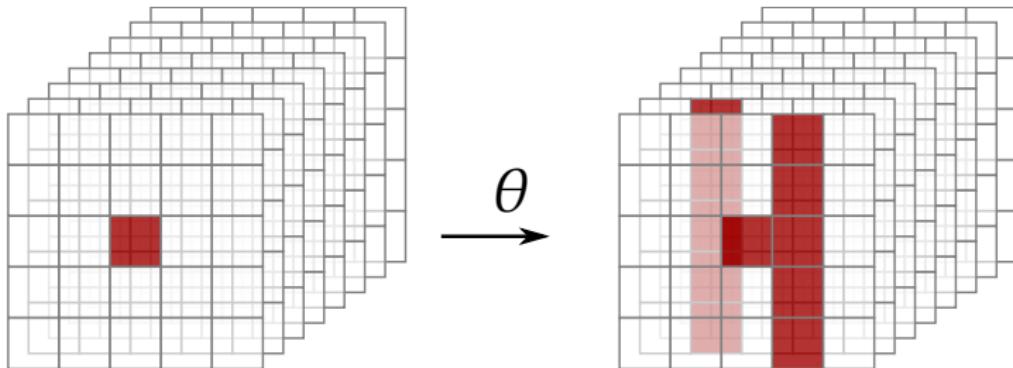
- Round function: $R = \iota \circ \pi \circ \rho \circ \theta' \circ \chi$
- Solves problem encountered before:



- π moves bits in same column to different columns!

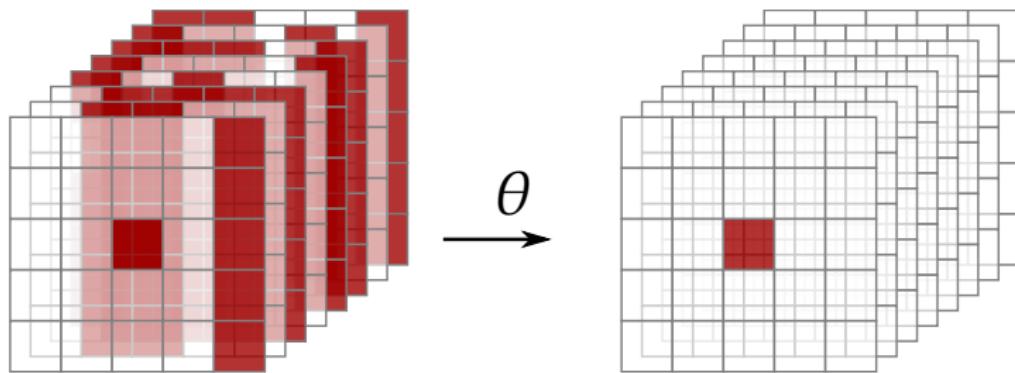
Almost there, still a final tweak ...

Tweaking θ' to θ



$$\begin{aligned} & 1 + (1 + y + y^2 + y^3 + y^4) (x + x^4 z) \\ & (\text{mod } \langle 1 + x^5, 1 + y^5, 1 + z^w \rangle) \end{aligned}$$

Inverse of θ



$$1 + (1 + y + y^2 + y^3 + y^4) \mathbf{Q},$$

$$\text{with } \mathbf{Q} = 1 + (1 + x + x^4 z)^{-1} \bmod \langle 1 + x^5, 1 + z^w \rangle$$

- \mathbf{Q} is dense, so:
 - Diffusion from single-bit output to input very high
 - Increases resistance against LC/DC and algebraic attacks

KECCAK-*f* summary

- Round function:

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

- Number of rounds: $12 + 2\ell$

- KECCAK-*f*[25] has 12 rounds
- KECCAK-*f*[1600] has 24 rounds

Outline

1 The SHA-3 competition

2 The sponge construction

3 Inside KECCAK

4 The SHA-3 FIPS

The long road to the SHA-3 FIPS

- February 2013: NIST-KECCAK-team meeting
 - SHA-2 replacement by now less urgent
 - ...but KECCAK is more than just hashing!
- NIST disseminates joint SHA-3 proposal
- Summer 2013: Snowden revelations
 - alleged NSA backdoor in DUAL EC DRBG
 - SHA-3 proposal framed as “NIST weakening KECCAK”
- Early 2014: standard takes shape addressing public concerns
- Friday, April 4, 2014: draft FIPS 202 for public comments
- August 2014: NIST announces plans at SHA-3 conference
- Mid 2015 (expected): FIPS 202 official publication

FIPS 202: what is inside?

- Content
 - KECCAK instances for
 - 4 hash functions
 - 2 XOFs
 - KECCAK- f all 7 block widths
 - even reduced-round versions
 - unlike AES FIPS that has only 1 of the 5 Rijndael widths
 - sponge construction
- Concept: toolbox for building other functions
 - tree hashing, MAC, encryption, ...
 - dedicated *special publications* (NIST SP 800-XX) under development

<http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/index.html>

XOF: eXtendable Output Function

“XOF: a function in which the output can be extended to any length.”

- Good for full domain hash, stream ciphers and key derivation
[Ray Perlner, SHA 3 workshop 2014]
- Quite natural for sponge
 - keeps state and delivers more output upon request
 - bits of output do not depend on the number of bits requested
- Allows simplification:
 - instead of separate hash functions per output length
 - a single XOF can cover **all** use cases:

$$\text{H-256}(M) = \lfloor \text{XOF}(M) \rfloor_{256}$$

MAC

Given a message and its hash code, as output by a cryptographic hash function, ensures that data has not been tampered with between the execution of the hash function and its verification, by recomputing the hash. However, using a hash function in this way requires the hash code itself to be protected in some way, by for example a digital signature, as otherwise the hash code itself could be tampered with. To avoid this problem MAC is introduced

$$\text{code} = \text{MAC}_k(m)$$

HMAC

$$HMAC = h(k||p_1||h(k||p_2||m))$$

CBC-MAC

$$m = m_1 || m_2 || \dots m_q,$$

$$l_1 = m_1, O_1 = \mathbf{Enc}_k(l_1),$$

$$l_i = m_i \oplus O_{i-1}, O_i = \mathbf{Enc}_k(l_i), \quad 2 \leq i \leq q,$$

$$MAC = \mathbf{Enc}_{k_1}(O_q).$$

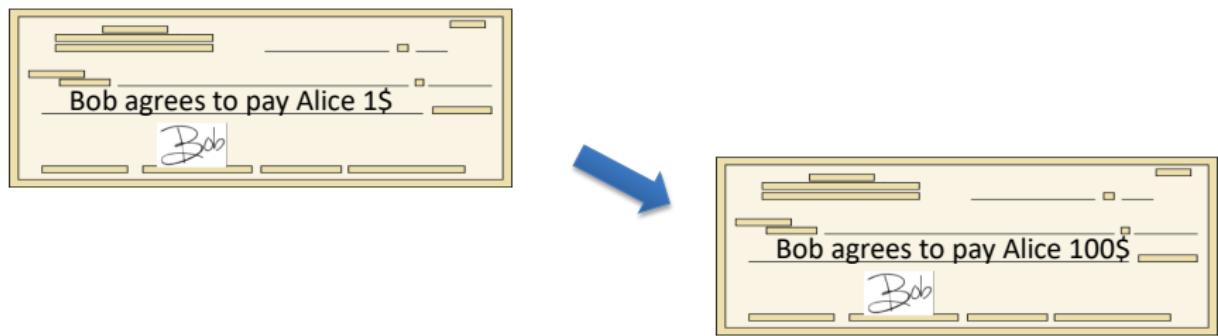
Digital Signatures, Certificates and PKI

Digital signatures

- control access to data
- allow users to authenticate themselves to a system
- allow users to authenticate data
- sign “real” documents
- NOT necessarily on a document
- NOT transferable to other documents
- NOT modifiable after they are made
- NOT produced by a person

Physical signatures

Goal: bind document to author

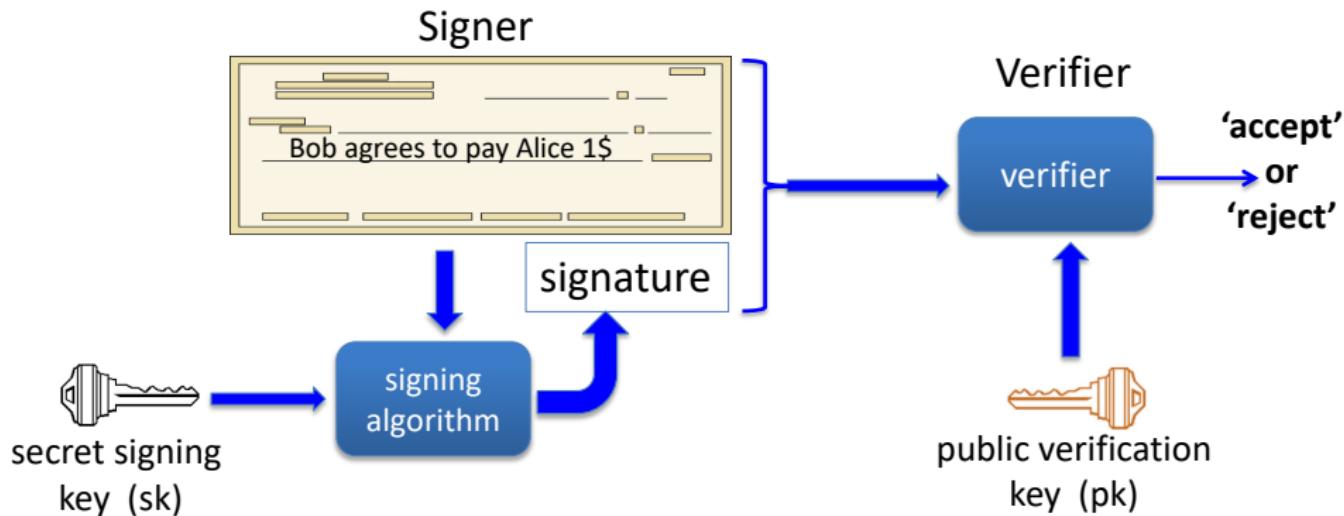


Problem in the digital world:

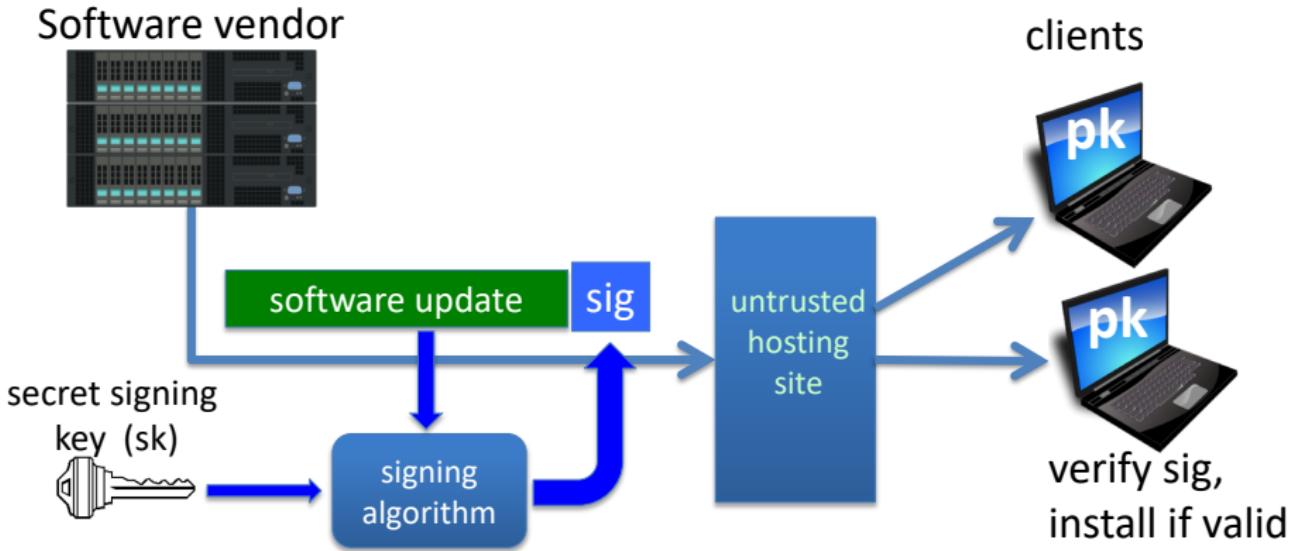
anyone can copy Bob's signature from one doc to another

Digital signatures

Solution: make signature depend on document



A more realistic example



Digital signatures: syntax

Def: a signature scheme $(\text{Gen}, \text{S}, \text{V})$ is a triple of algorithms:

- $\text{Gen}()$: randomized alg. outputs a key pair (pk, sk)
- $\text{S}(\text{sk}, m \in M)$ outputs sig. σ
- $\text{V}(\text{pk}, m, \sigma)$ outputs ‘accept’ or ‘reject’

Consistency: for all (pk, sk) output by Gen :

$$\forall m \in M: \text{V}(\text{pk}, m, \text{S}(\text{sk}, m)) = \text{'accept'}$$

Digital signatures: security

Attacker's power: **chosen message attack**

- for m_1, m_2, \dots, m_q attacker is given $\sigma_i \leftarrow S(\text{sk}, m_i)$

Attacker's goal: **existential forgery**

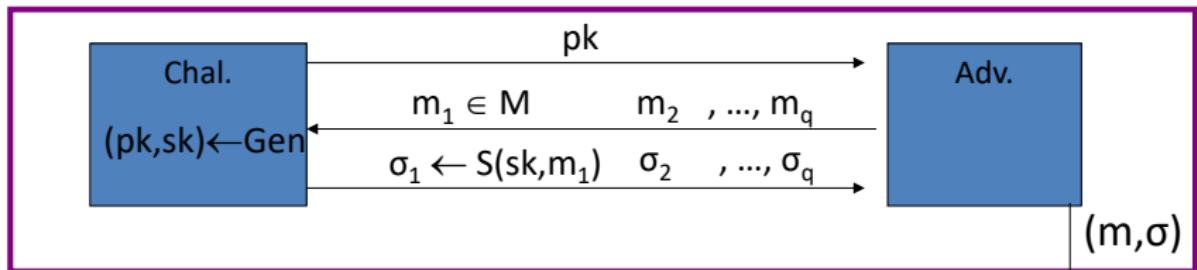
- produce some new valid message/sig pair (m, σ) .

$$m \notin \{m_1, \dots, m_q\}$$

⇒ attacker cannot produce a valid sig. for a new message

Secure signatures

For a sig. scheme $(\text{Gen}, \text{S}, \text{V})$ and adv. A define a game as:



Adv. wins if $\text{V}(\text{pk}, \text{m}, \sigma) = \text{'accept'}$ and $\text{m} \notin \{\text{m}_1, \dots, \text{m}_q\}$

Def: $\text{SS} = (\text{Gen}, \text{S}, \text{V})$ is **secure** if for all “efficient” A :

$$\text{Adv}_{\text{SIG}}[A, \text{SS}] = \Pr[\text{A wins}] \text{ is “negligible”}$$

Let $(\text{Gen}, \text{S}, \text{V})$ be a signature scheme.

Suppose an attacker is able to find $m_0 \neq m_1$ such that

$$\text{V}(\text{pk}, m_0, \sigma) = \text{V}(\text{pk}, m_1, \sigma) \quad \text{for all } \sigma \text{ and keys } (\text{pk}, \text{sk}) \leftarrow \text{Gen}$$

Can this signature be secure?

- Yes, the attacker cannot forge a signature for either m_0 or m_1
- No, signatures can be forged using a chosen msg attack
- It depends on the details of the scheme

Alice generates a (pk, sk) and gives pk to her bank.

Later Bob shows the bank a message $m = \text{"pay Bob 100\$"}$
properly signed by Alice, i.e. $V(pk, m, sig) = \text{'yes'}$

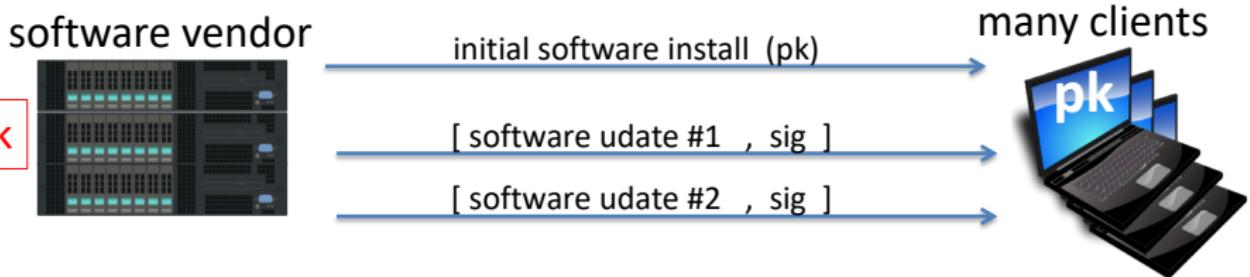
Alice says she never signed m . Is Alice lying?

- Alice is lying: existential unforgeability means Alice signed m and therefore the Bank should give Bob 100\$ from Alice's account
- Bob could have stolen Alice's signing key and therefore the bank should not honor the statement
- What a mess: the bank will need to refer the issue to the courts

Applications

Code signing:

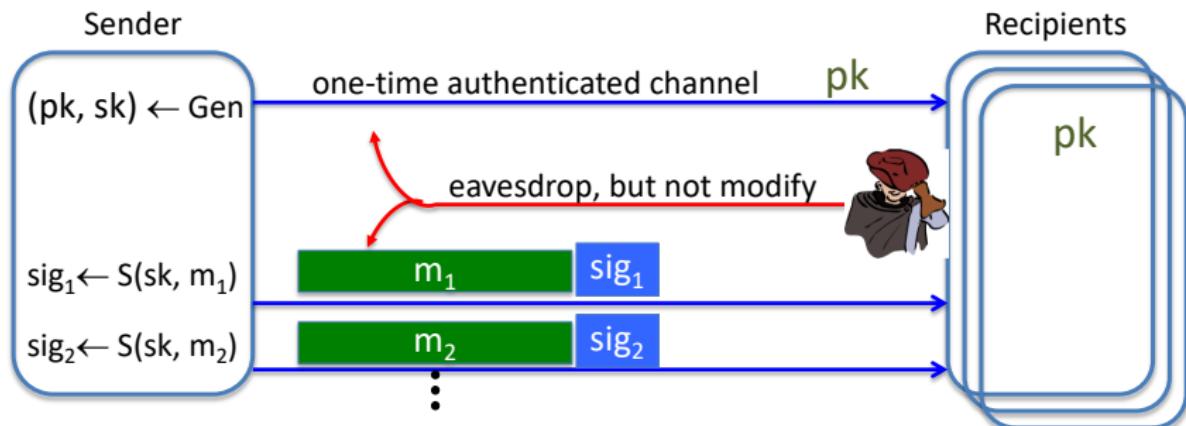
- Software vendor signs code
- Clients have vendor's pk. Install software if signature verifies.



More generally:

One-time authenticated channel (non-private, one-directional)
⇒ many-time authenticated channel

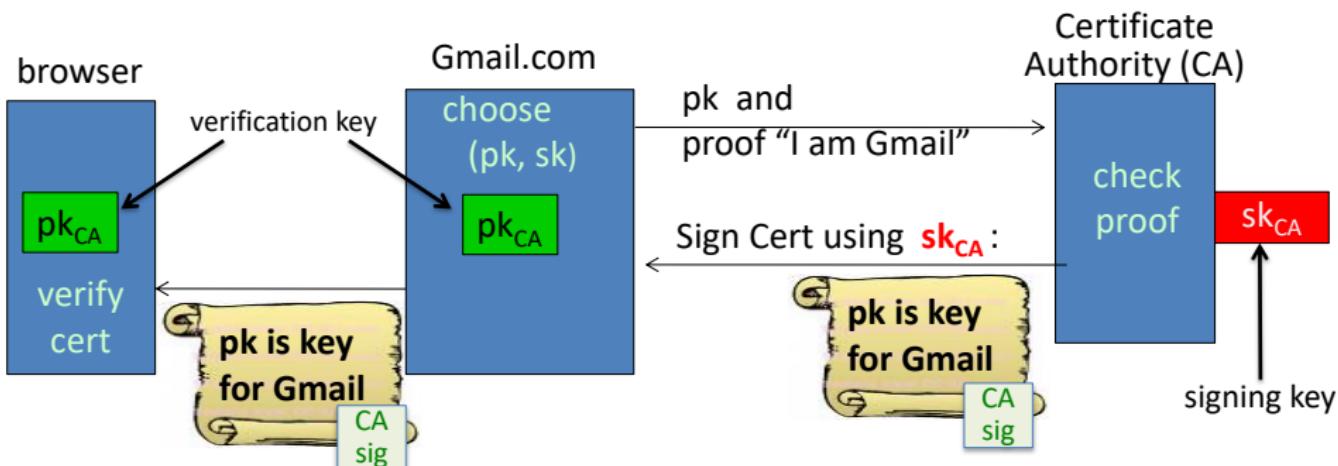
Initial software install is authenticated, but not private



Important application: Certificates

Problem: browser needs server's public-key to setup a session key

Solution: server asks trusted 3rd party (CA) to sign its public-key pk



Server uses Cert for an extended period (e.g. one year)

Certificates: example

Important fields:

Serial Number	5814744488373890497	←
Version	3	
Signature Algorithm	SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)	
Parameters	none	
Not Valid Before	Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time	
Not Valid After	Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time	
Public Key Info		
Algorithm	Elliptic Curve Public Key (1.2.840.10045.2.1)	
Parameters	Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)	
Public Key	65 bytes : 04 71 6C DD E0 0A C9 76 ...	←
Key Size	256 bits	
Key Usage	Encrypt, Verify, Derive	
Signature	256 bytes : 8A 38 FE D6 F5 E7 F6 59 ...	←

Equifax Secure Certificate Authority
↳ GeoTrust Global CA
↳ Google Internet Authority G2
↳ mail.google.com

 mail.google.com

Issued by: Google Internet Authority G2
Expires: Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time
This certificate is valid

▼ Details

Subject Name		
Country	US	
State/Province	California	
Locality	Mountain View	
Organization	Google Inc	
Common Name	mail.google.com	←
Issuer Name		
Country	US	
Organization	Google Inc	
Common Name	Google Internet Authority G2	

What entity generates the CA's secret key sk_{CA} ?

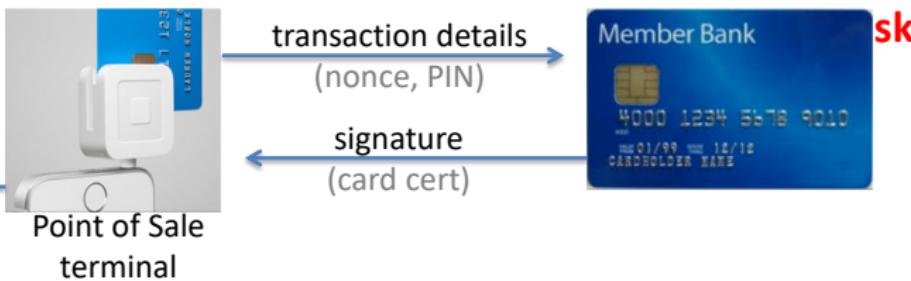
- the browser
- Gmail
- the CA
- the NSA

Applications with few verifiers

EMV payments:

(greatly simplified)

transaction details
and signature



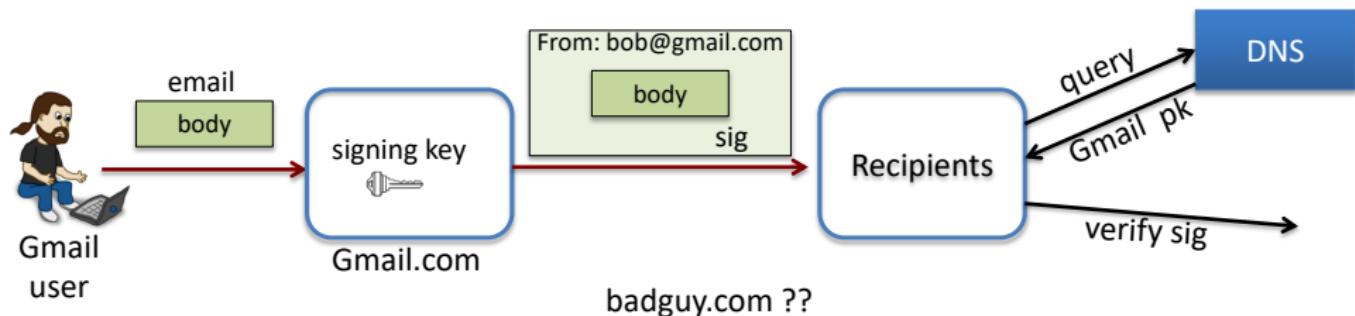
Signed email: sender signs email it sends to recipients

- Every recipient has sender's public-key (and cert).
A recipient accepts incoming email if signature verifies.

Signing email: DKIM (domain key identified mail)

Problem: bad email claiming to be from someuser@gmail.com
but in reality, mail is coming from domain **baguy.com**
⇒ Incorrectly makes gmail.com look like a bad source of email

Solution: **gmail.com** (and other sites) sign every outgoing mail



example DKIM header from gmail.com

X-Google-DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
d=1e100.net; s=20130820; (lookup 20130820._domainkey.1e100.net in DNS for public key)
h=x-gm-message-state:mime-version:in-reply-to:references:from:date:
message-id:subject:to:content-type;
bh=MDr/xwte+/JQSgCG+T2R2Uy+SuTK4/gxqdxMc273hPQ=; (hash of message body)

b=dOTpUVoAcrWS6AzmcPMreo09G9viS+sn1z6g+GpC/ArkfMEMcffOJ1s9u5Xa5KC+6K
XRzwZhAWYqFr2a0ywCjbGECBPIE5ccOi9DwMjnvJRYEwNk7/sMzFfx+0L3nTqgTyd0ED
EGWdN3upzSXwBrXo82wVcRRCnQ1yUITddnHgEoEFg5WV37DRP/eq/hOB6zFNTRBwkvfS
0tC/DNdRwfspO+UboRU2eiWaqJWPjxL/abS7xA/q1VGz0ZoI0y3/SCkxdg4H80c61DU
jdVYhCUd+dSV5fISouLQT/q5DYEjlNQbi+EcbL00liu4o623SDEeyx2isUgcvii2VxTWQ
m80Q==

Gmail's signature on headers, including DKIM header (2048 bits)

Suppose recipients could retrieve new data from DNS for every email received, could Gmail implement DKIM without signatures?
(ignoring, for now, the increased load on the DNS system)

- Yes, Gmail would write to DNS a collision-resistant hash of every outgoing email. The recipient retrieves the hash from DNS and compares to the hash of the incoming message.
- No, the proposal above is insecure.

⇒ Signatures reduce the frequency that recipients need to query DNS

Applications: summary

- Code signing
- Certificates
- Signed email (e.g. DKIM)
- Credit-card payments: EMV

and many more.

When to use signatures

Generally speaking:

- If one party signs and one party verifies: **use a MAC**
 - Often requires interaction to generate a shared key
 - Recipient can modify the data and re-sign it before passing the data to a 3rd party
- If one party signs and many parties verify: **use a signature**
 - Recipients **cannot** modify received data before passing data to a 3rd party (non-repudiation)

Review: three approaches to data integrity

1. **Collision resistant hashing**: need a read-only public space



2. **Digital signatures**: vendor must manage a long-term secret key

- Vendor's signature on software is shipped with software
- Software can be downloaded from an untrusted distribution site

3. **MACs**: vendor must compute a new MAC of software for every client

- and must manage a long-term secret key (to generate a per-client MAC key)

Review: digital signatures

Def: a signature scheme $(\text{Gen}, \text{S}, \text{V})$ is a triple of algorithms:

- $\text{Gen}()$: randomized alg. outputs a key pair (pk, sk)
- $\text{S}(\text{sk}, m \in M)$ outputs sig. σ
- $\text{V}(\text{pk}, m, \sigma)$ outputs ‘yes’ or ‘no’

Security:

- Attacker’s power: chosen message attack
- Attacker’s goal: existential forgery

Extending the domain with CRHF

Let $\mathbf{Sig} = (\text{Gen}, \text{S}, \text{V})$ be a sig scheme for short messages, say $M = \{0,1\}^{256}$

Let $H: M^{\text{big}} \rightarrow M$ be a hash function (s.g. SHA-256)

Def: $\mathbf{Sig}^{\text{big}} = (\text{Gen}, \text{S}^{\text{big}}, \text{V}^{\text{big}})$ for messages in M^{big} as:

$$\text{S}^{\text{big}}(\text{sk}, \text{m}) = \text{S}(\text{sk}, H(\text{m})) ; \quad \text{V}^{\text{big}}(\text{pk}, \text{m}, \sigma) = \text{V}(\text{pk}, H(\text{m}), \sigma)$$

Thm: If \mathbf{Sig} is a secure sig scheme for M and H is collision resistant
then $\mathbf{Sig}^{\text{big}}$ is a secure sig scheme for M^{big}

⇒ suffices to construct signatures for short 256-bit messages

Suppose an attacker finds two distinct messages m_0, m_1
such that $H(m_0) = H(m_1)$. Can she use this to break **Sig^{big}** ?

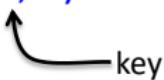
- No, **Sig^{big}** is secure because the underlying scheme **Sig** is
- It depends on what underlying scheme **Sig** is used
- Yes, she would ask for a signature on m_0 and obtain an existential forgery for m_1

Primitives that imply signatures: OWF

Recall: $f: X \rightarrow Y$ is a **one-way function** (OWF) if:

- easy: for all $x \in X$ compute $f(x)$
- inverting f is hard:

Example: $f(x) = \text{AES}(x, 0)$



Signatures from OWF: Lamport-Merkle (see next module), Rompel

- Signatures are long:
 - stateless $\Rightarrow > 40\text{KB}$
 - stateful $\Rightarrow > 4\text{KB}$

Primitives that imply signatures: TDP

Recall: $f: X \rightarrow X$ is a **trapdoor permutation** (TDP) if:

- easy: for all $x \in X$ compute $f(x)$
- inverting f is hard, **unless one has a trapdoor**

Example: RSA

Signatures from TDP: very simple and practical (next segment)

- Commonly used for signing certificates

Primitives that imply signatures: DLOG

$\mathbf{G} = \{1, g, g^2, \dots, g^{q-1}\}$: finite cyclic group with generator g , $|G| = q$

discrete-log in G is hard if $f(x) = g^x$ is a one-way function

- note: $f(x+y) = f(x) \cdot f(y)$

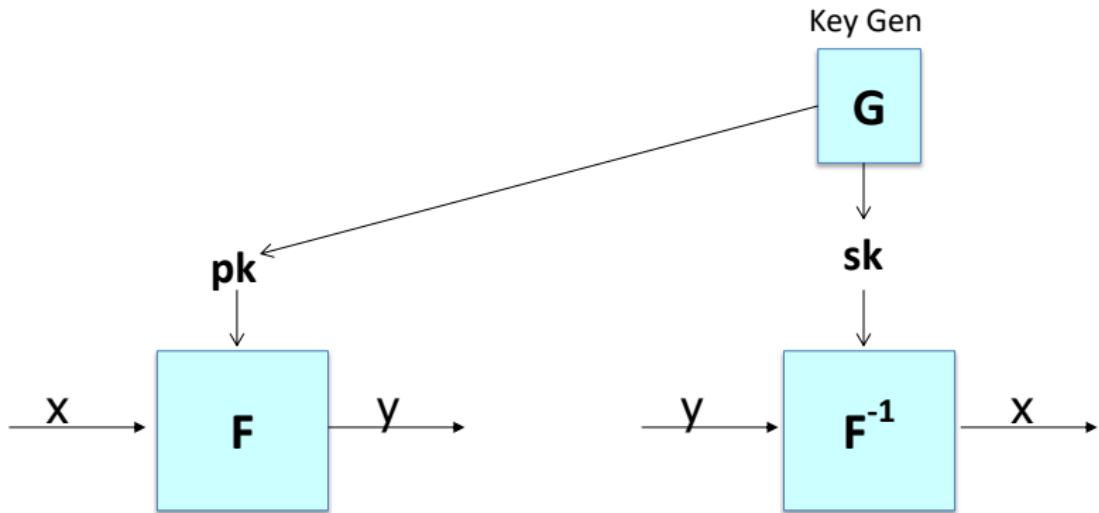
Examples: \mathbb{Z}_p^* = (multiplication mod p) for a large prime p

$E_{a,b}(\mathbb{F}_p)$ = (group of points on an elliptic curve mod p)

Signatures from DLOG: ElGamal, Schnorr, DSA, EC-DSA, ...

- Will construct these signatures in week 3

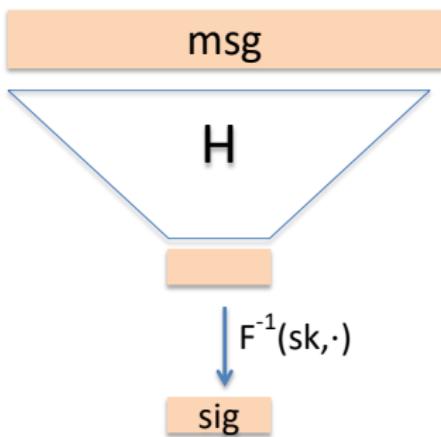
Review: Trapdoor permutation (G, F, F^{-1})



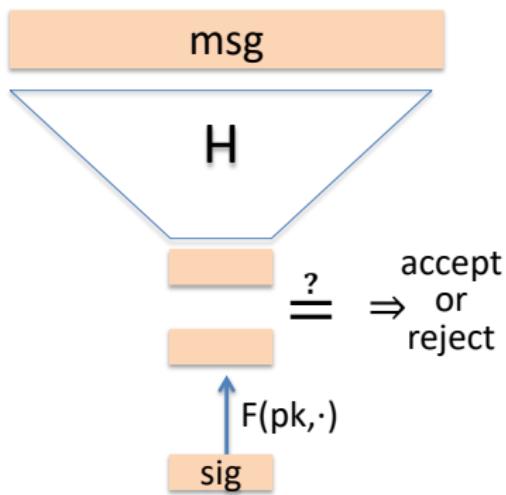
$f(x) = F(pk, x)$ is one-to-one ($X \rightarrow X$) and is a **one-way function**.

Full Domain Hash Signatures: pictures

$S(sk, \text{msg})$:



$V(pk, \text{msg}, \text{sig})$:



Full Domain Hash (FDH) Signatures

(G_{TDP}, F, F^{-1}) : Trapdoor permutation on domain X

$H: M \rightarrow X$ hash function (FDH)

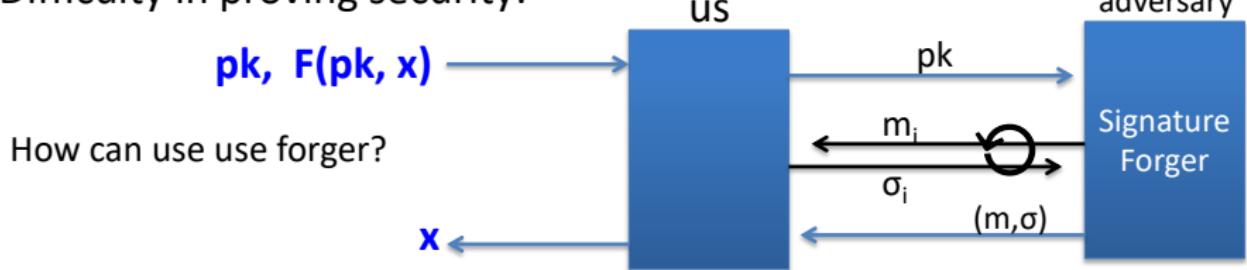
(Gen, S, V) signature scheme:

- Gen : run G_{TDP} and output pk, sk
- $S(sk, m \in M)$: output $\sigma \leftarrow F^{-1}(sk, H(m))$
- $V(pk, m, \sigma)$: output $\begin{cases} \text{'accept'} & \text{if } F(pk, \sigma) = H(m) \\ \text{'reject'} & \text{otherwise} \end{cases}$

Security

Thm [BR]: (G_{TDP}, F, F^{-1}) secure TDP $\Rightarrow (Gen, S, V)$ secure signature
when $H: M \rightarrow X$ is modeled as an “ideal” hash function

Difficulty in proving security:



How can we use a forger?

Solution: “we” will know sig. on **all-but-one** of m where adv. queries H().
Hope adversary gives forgery for that single message.

Why hash the message?

Suppose we define NoHash-FDH as:

- $S'(sk, m \in X)$: output $\sigma \leftarrow F^{-1}(sk, m)$
- $V'(pk, m, \sigma)$: output ‘accept’ if $F(pk, \sigma) = m$

Is this scheme secure?

- Yes, it is not much different than FDH
- No, for any $\sigma \in X$, σ is a signature forgery for the msg $m = F(pk, \sigma)$
- Yes, the security proof for FDH applies here too
- It depends on the underlying TDP being used

RSA-FDH

Gen: generate an RSA modulus $N = p \cdot q$ and $e \cdot d = 1 \pmod{\phi(N)}$
construct CRHF $H: M \rightarrow \mathbb{Z}_N$
output $pk = (N, e, H)$, $sk = (N, d, H)$

- **S($sk, m \in M$):** output $\sigma \leftarrow H(m)^d \pmod{N}$
- **V(pk, m, σ):** output ‘accept’ if $H(m) = \sigma^e \pmod{N}$

Problem: having H depend on N is slightly inconvenient

PKCS1 v1.5 signatures

RSA trapdoor permutation: $\text{pk} = (\text{N}, \text{e})$, $\text{sk} = (\text{N}, \text{d})$

- $S(\text{sk}, m \in M)$:



output: $\sigma \leftarrow (\text{EM})^d \bmod N$

- $V(\text{pk}, m \in M, \sigma)$: verify that $\sigma^e \bmod N$ has the correct format

Security: no security analysis, not even with ideal hash functions

RSA signatures in practice often use $e=65537$ (and a large d). As a result, sig verification is $\approx 20x$ faster than sig generation.

$e=3$ gives even faster signature verification.

Suppose an attacker finds an $m^* \in M$ such that

EM is a perfect cube (e.g. $8=2^3$, $27=3^3$, $64=4^3$).

Can she use this m^* to break PKCS1?

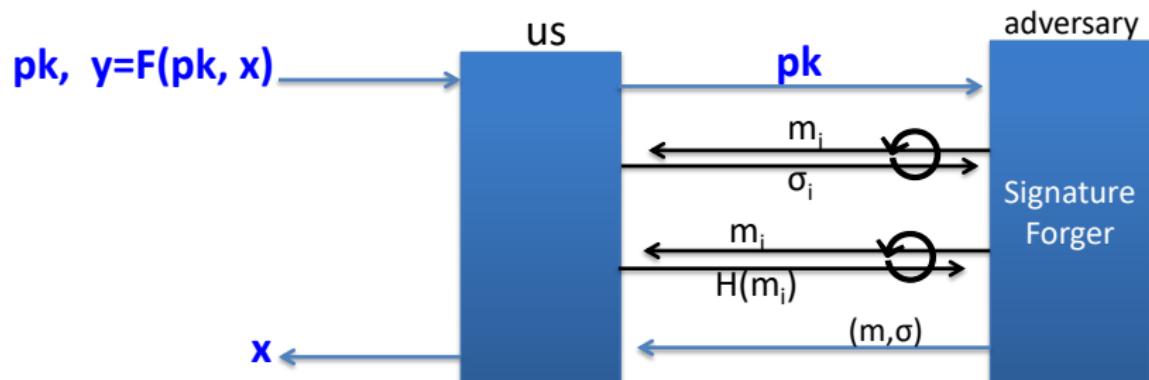
- Yes, the cube root of EM (over the integers) is a sig. forgery for m^*
- No, this has no impact on PKCS1 signatures
- Yes, but the attack only works for a few 2048-bit moduli N
- It depends on what hash function is begin used

Proving security of RSA-FDH

(G, F, F^{-1}) : secure TDP with domain X

Recall FDH sigs: $S(\text{sk}, m) = F^{-1}(\text{sk}, H(m))$ where $H: M \rightarrow X$

We will show: TDP is secure \Rightarrow FDH is secure, when H is a random function

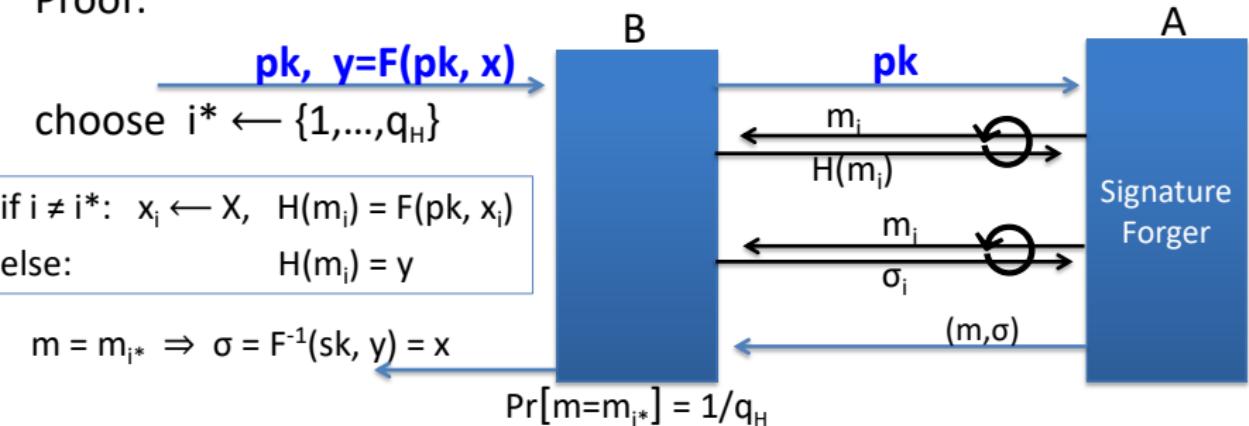


Proving security

Thm [BR]: (G_{TDP}, F, F^{-1}) secure TDP $\Rightarrow (G_{TDP}, S, V)$ secure signature
when $H: M \rightarrow X$ is modeled as a random oracle.

$$\forall A \exists B: \text{Adv}_{\text{SIG}}^{(\text{RO})}[A, \text{FDH}] \leq q_H \cdot \text{Adv}_{\text{TDP}}[B, F]$$

Proof:



Proving security

Thm [BR]: (G_{TDP}, F, F^{-1}) secure TDP $\Rightarrow (G_{TDP}, S, V)$ secure signature
when $H: M \rightarrow X$ is modeled as a random oracle.

$$\forall A \exists B: \text{Adv}_{\text{SIG}}^{(\text{RO})}[A, \text{FDH}] \leq q_H \cdot \text{Adv}_{\text{TDP}}[B, F]$$

Proof:



$$\text{So: } \underbrace{\text{Adv}_{\text{TDP}}[B, F]}_{\substack{\text{Prob. B} \\ \text{outputs } x}} \geq \underbrace{(1/q_H)}_{\Pr[m=m_{i^*}]} \cdot \underbrace{\text{Adv}_{\text{SIG}}[A, \text{FDH}]}_{\substack{\text{Prob. forger A} \\ \text{outputs valid forgery}}}$$

Alg. B has table:

$$m_1, x_1 : H(m_1) = F(pk, x_1)$$

$$m_2, x_2 : H(m_2) = F(pk, x_2)$$

⋮

$$m_{i^*}, \quad H(m_{i^*}) = y$$

⋮

$$m_q, x_q : H(m_q) = F(pk, x_q)$$

How B answers a signature query m_i :

Partial domain hash:

Suppose (G_{TDP}, F, F^{-1}) is defined over domain $X = \{0, \dots, B-1\}$
but $H: M \rightarrow \{0, \dots, B/2\}$.

Can we prove FDH secure with such an H ?

- No, FDH is only secure with a full domain hash
- Yes, but we would need to adjust how B defines $H(m_i)$ in the proof
- It depends on what TDP is used

PSS: Tighter security proof

Some variants of FDH:

tight reduction from forger to inverting the TDP (no q_H factor).
Still assuming hash function H is “ideal.”

Examples:

- PSS [BR'96]: part of the PKCS1 v2.1 standard
- KW'03: $S((sk, k), m) = [b \leftarrow PRF(k, m) \in \{0,1\} , F^{-1}(sk, H(b || m))]$
- many others

A new tool: pairings

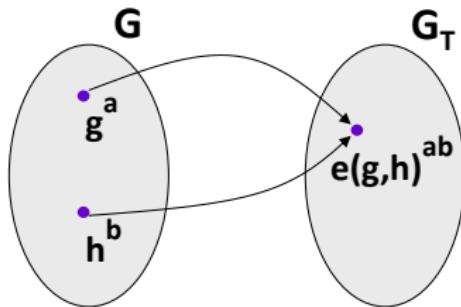
Secure signature without “ideal” hash function (a.k.a. random oracles):

- can be built from RSA, but
- most efficient constructions use **pairings**

G, G_T : finite cyclic groups $G = \{1, g, \dots, g^{p-1}\}$

Def: A **pairing** $e: G \times G \rightarrow G_T$ is a map:

- bilinear: $e(g^a, h^b) = e(g, h)^{ab} \quad \forall a, b \in \mathbb{Z}, g, h \in G$
- efficiently computable and non-degenerate:
 g generates $G \Rightarrow e(g, g)$ generates G_T



BLS: a simple signature from pairings

$e: G \times G \rightarrow G_T$ a pairing where $|G|=p$, $g \in G$ generator, $H: M \rightarrow G$

Gen: $sk = (\text{random } \alpha \text{ in } Z_p) , \ pk = g^\alpha \in G$

$S(sk, m)$: output $\sigma = H(m)^\alpha \in G$

$V(pk, m, \sigma)$: accept if $e(g, \sigma) \stackrel{?}{=} e(pk, H(m))$

Thm: secure assuming CDH in G is hard, when H is a random oracle

Security without random oracles [BB'04]

Gen: $\text{sk} = (\text{rand. } \alpha, \beta \leftarrow \mathbb{Z}_p)$, $\text{pk} = (g, y=g^\alpha \in G, z=g^\beta \in G)$

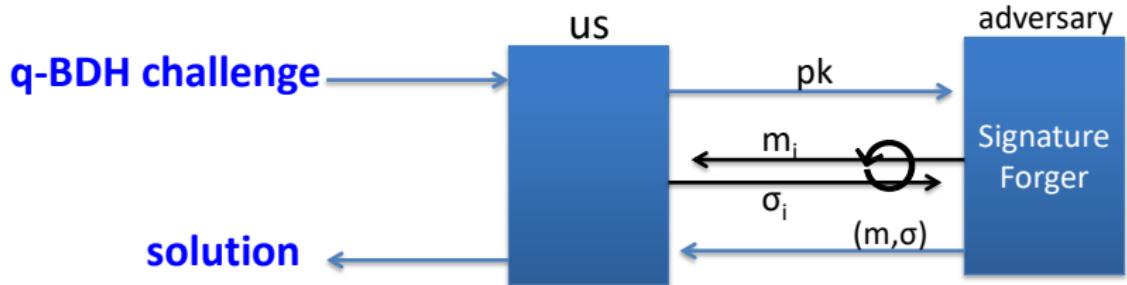
$S(\text{sk}, m \in \mathbb{Z}_p)$: $r \leftarrow \mathbb{Z}_p$, $\sigma = g^{1/(\alpha+r\beta+m)} \in G$, output (r, σ)

$V(\text{pk}, m, (r, \sigma))$: accept if $e(\sigma, y \cdot z^r \cdot g^m) \stackrel{?}{=} e(g, g)$

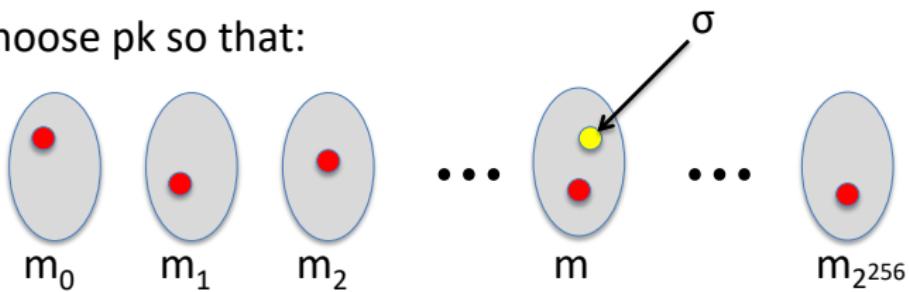
Thm: secure assuming q_s -BDH in G is hard

$$\forall A \exists B : \text{Adv}_{\text{SIG}}[A, \text{BBsig}] \leq \text{Adv}_{q_s\text{-BDH}}[B, G] + (q_s/p)$$

Proof strategy



We choose pk so that:



Signature lengths

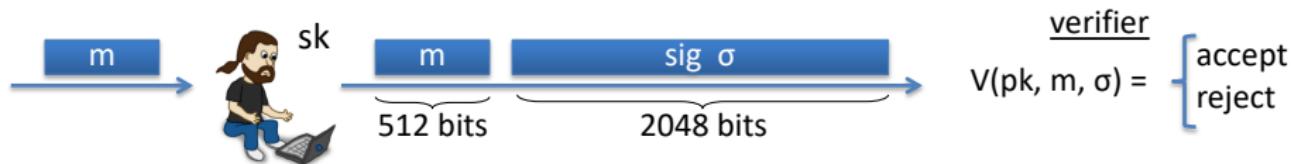
Goal: best existential forgery attack time $\geq 2^{128}$

<u>algorithm</u>	<u>signature size</u>
RSA	2048-3072 bits
EC-DSA	512 bits
Schnorr	384 bits
BLS	256 bits

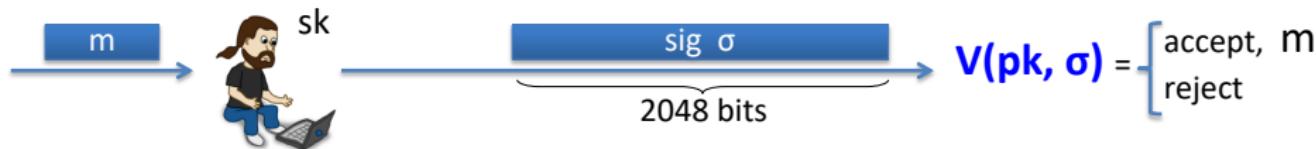
Open problem: practical 128-bit signatures

Signatures with Message Recovery

Suppose Alice needs to sign a short message, say $m \in \{0,1\}^{512}$



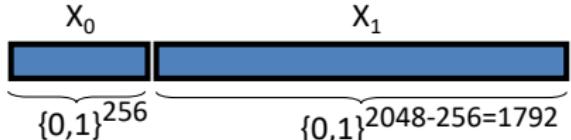
Can we do better? Yes: signatures with message recovery



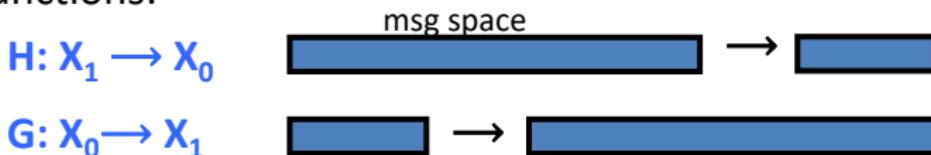
Security: existential unforgeability under a chosen message attack

Sigs with Message Recovery: Example

(G_{TDP}, F, F^{-1}) : TDP on domain $(X_0 \times X_1)$



Hash functions:



Signing: $S(\text{sk}, m \in X_1): h \leftarrow H(m) \in X_0$

$EM = \underbrace{h}_{256 \text{ bits}} \quad m \oplus G(h) \in X_0 \times X_1$

output: $\sigma \leftarrow F^{-1}(\text{sk}, EM)$

Sigs with Message Recovery: Example

$S(\text{sk}, m \in X_1)$: choose random $h \leftarrow H(m) \in X_0$

$EM = \boxed{\begin{array}{|c|c|} \hline h & m \oplus G(h) \\ \hline \end{array}} \in X_0 \times X_1$

256 bits

output: $\sigma \leftarrow F^{-1}(\text{sk}, EM)$

$V(\text{pk}, \sigma)$: $(x_0, x_1) \leftarrow F(\text{pk}, \sigma), m \leftarrow x_1 \oplus G(x_0)$
if $x_0 = H(m)$ output “accept, m ” else “reject”

Thm: (G_{TDP}, F, F^{-1}) secure TDP $\Rightarrow (G_{TDP}, S, V)$ secure MR signature
when H, G are modeled as random oracles

Standard for sigs with message-recovery: **RSA-PSS-R** (PKCS1)

Consider the following MR signature: $S(\text{sk}, m) = F^{-1}(\text{sk}, [m \parallel H(m)])$

$V(\text{pk}, \sigma)$: $(m, h) \leftarrow F(\text{pk}, \sigma)$
if $h = H(m)$ outputs "accept, m "

Unfortunately, we can't prove security.

Should we use this scheme with RSA and with H as SHA-256?

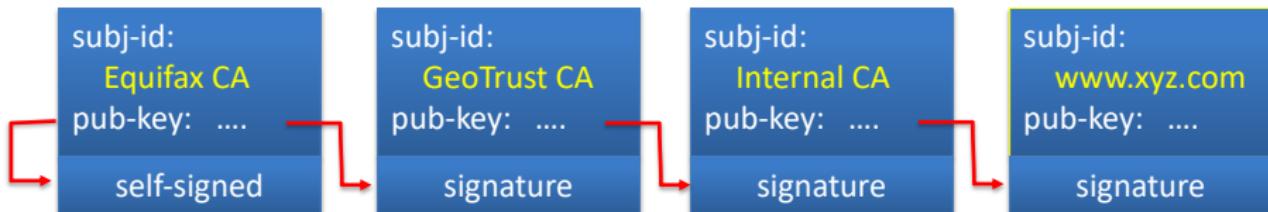
(ISO/IEC 9796-2 sigs. and EMV sigs.)

- Yes, unless someone discovers an attack
- No, only use schemes that have a clear security analysis
- It depends on the size of the RSA modulus

Aggregate Signatures

[BGLS'03]

Certificate chain:



Aggregate sigs: lets anyone compress n signatures into one

$$\begin{array}{l} \mathsf{pk}_1, m_1 \rightarrow \sigma_1 \\ \vdots \\ \mathsf{pk}_n, m_n \rightarrow \sigma_n \end{array} \xrightarrow{\text{aggregate}} \sigma^*$$

$V_{\text{agg}}(\bar{\mathsf{pk}}, \bar{m}, \sigma^*) = \text{"accept"}$
means for $i=1, \dots, n$:
user i signed msg m_i

Aggregate Signatures

[BGLS'03]

Certificate chain with aggregates sigs:

subj-id:
Equifax CA
pub-key:

subj-id:
GeoTrust CA
pub-key:

subj-id:
Internal CA
pub-key:

subj-id:
www.xyz.com
pub-key:
aggregate-sig

Aggregate sigs: let us compress n signatures into one

$$\begin{array}{l} \mathsf{pk}_1, m_1 \rightarrow \sigma_1 \\ \vdots \\ \mathsf{pk}_n, m_n \rightarrow \sigma_n \end{array} \xrightarrow{\text{aggregate}} \sigma^*$$

$V_{\text{agg}}(\bar{\mathsf{pk}}, \bar{\mathbf{m}}, \sigma^*) = \text{"accept"}$
means for $i=1,\dots,n$:
user i signed msg m_i

CA based key distribution

- All users have a trusted copy of the public key of the CA. For example these come embedded in your browser when you buy your computer, and you 'of course' trust the vendor of the computer and the manufacturer of the software on your computer.
- The CA's job is to digitally sign data strings containing the following information (Alice, Alice's public key). This data string, and the associated signature is called a digital certificate. The CA will only sign this data if it truly believes that the public key really does belong to Alice.
- When Alice now sends you her public key, contained in a digital certificate, you now trust that the purported key really is that of Alice, since you trust the CA to do its job correctly.

Outline

- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate Revocation
- **PKI - Public Key Infrastructure**
 - PKI Standardization and PKIX Management

Outline

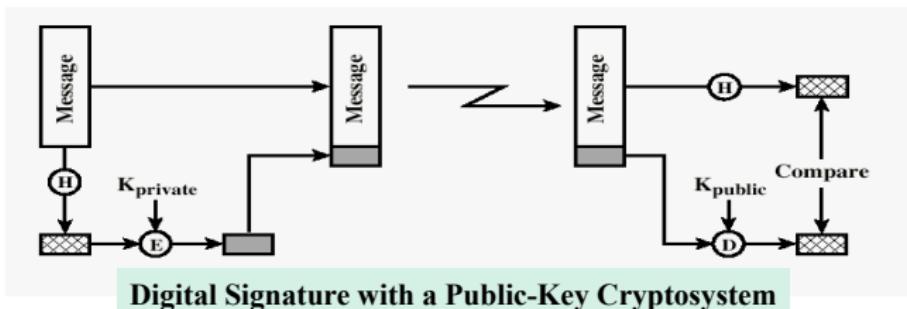
- 
- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate revocation
 - **PKI - Public Key Infrastructure**
 - PKI Standardization and PKIX Management

X.509 Authentication Service

- Distributed set of servers that maintains a database of certificates.
 - Running by certification authorities (CA)
 - acting as issuers of certificates (PK certificates)
- Each certificate contains:
 - The public key of a distinguished subject name (principal, user) association
 - Subject name, Subject's public key information fields
 - Other attributes with additional information as a list of other (field, value) pairs
 - Issuer UID, serial number, version, validity, relevant information of cipher-suites used, verification control, policy extensions,
 - Signed with the private key of a CA.
 - Digital signature covering all the other fields
 - Hash of fields, signed with the CA private key

X.509 Authentication Service

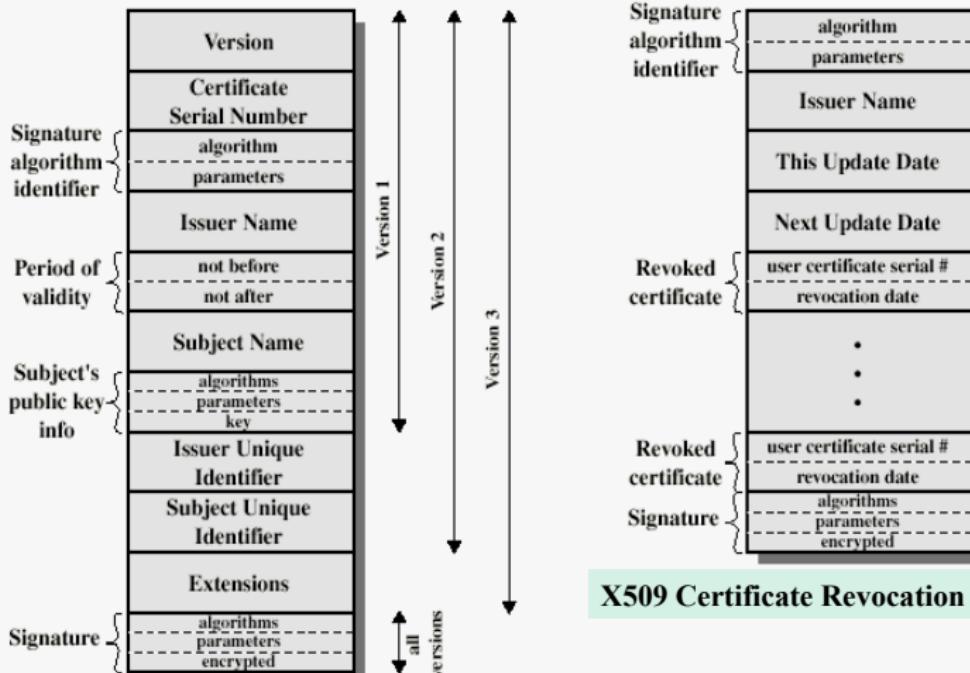
- Is used in S/MIME, IP Security, SSL/TLS, SET and many other security services, applications, commercial systems and other security standards
- Public key digital signatures recommended
 - RSA
 - DSA (or DSS)



Outline

- 
- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate revocation
 - **PKI - Public Key Infrastructure**
 - PKI Standardization and PKIX Management

X.509 Certificate and CRL Formats



X.509 certificate (fields in different versions)

X.509 Certificate Revocation List

X.509 Notation

- Standard notation in the standard:

$CA <<A>> = \{A, V, SN, AI, CA, TA, KpubA\}_{SigCA}$

$Y <<X>>$

represents a certificate of entity X issued by the certification authority Y

$Y \{I\}$ or $\{I\}_{SigY}$

Represents I signed by Y, with a digital public-key signature scheme (PK signature standard)

Verification of certificates => imply that the verifiers previously obtained, in a trusted way, the CA public key

- Or trust based on Certification Chains

Obtaining a User's Certificate

- Characteristics of certificates generated by CAs:
 - Any user with access to the public key of the CA can recover and validate the user public key that was certified (by a direct or reverse trust certification chain verification)
 - Users can exchange certificates and certification chains for verification
 - No part other than the CA can issue and modify the certificate, without this being detected.
 - Certificates are unforgeable (same property as defined for digital signatures), so it is possible to place them in public directories or repositories

More extensible TCB model

- Different entities involved, acting with different roles in a distributed way:
 - Difference between:
 - CA Certification authorities (*Cert. ISSUING*)
 - Different level CAs: aggregated in a direct certification chain
 - » Root CA, Level 2 CA, Level 3 CA, etc
 - » Model practically used in "well-known CA companies" or "CA delegation companies"
 - RA Registration authorities (*REGISTRATION, ENROLLMENT DELEGATION*)
 - CRL Issuers (*CRL ISSUERS*)
 - CRs or Certification Repositories (*DISTRIBUTION, On demand REQUEST-REPLY*)

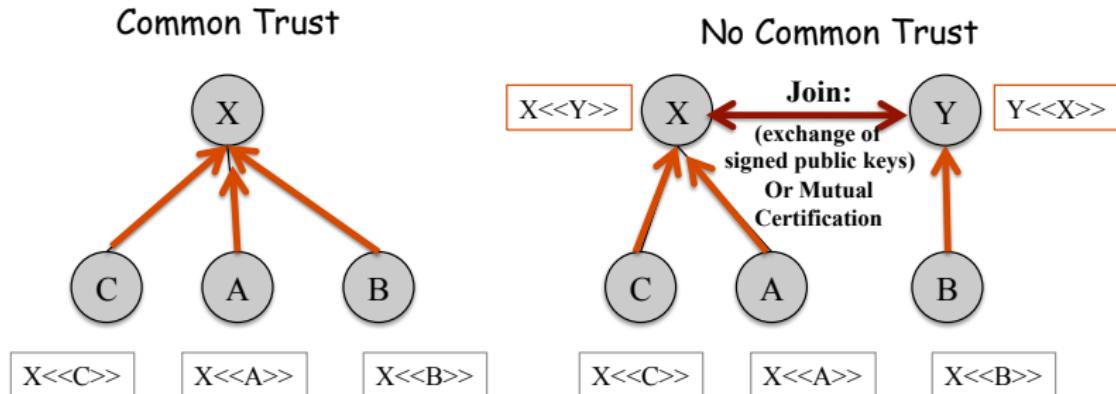
Outline

- 
- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate Revocation
 - **PKI - Public Key Infrastructure**
 - PKI Standardization and PKIX Management

Trust and validation chains

- **Common trust based Validation**
 - When all users subscribe to the same CA
 - Model for a small community of users (non-scalable, centralized-root trust)
 - Any user A can transmit directly the certificate to any other
 - for message authentication
 - or eventually for key or security associations establishment and message confidentiality
- **No common trust verification conditions**
 - Model for a large community of users (scalable model)
 - Users need to have Public Keys of all the CAs
 - Or it may be more practical to consider that
 - There will be several CAs,
 - But each of which securely provides its public key to some fraction of the users (cross-certification links in the certification hierarchy)

Solution for no common trust



- A obtains **X<<Y>>** from a directory
- A obtains **Y<>** from a directory (or directly from B)
- A uses the chain **Y <>, X<<Y>>**
B can use the chain: **X <<A>> Y<<X>>**
or reverse chain X<<A>> X<<Y>>
- Possible generalization for long paths (join at higher level)

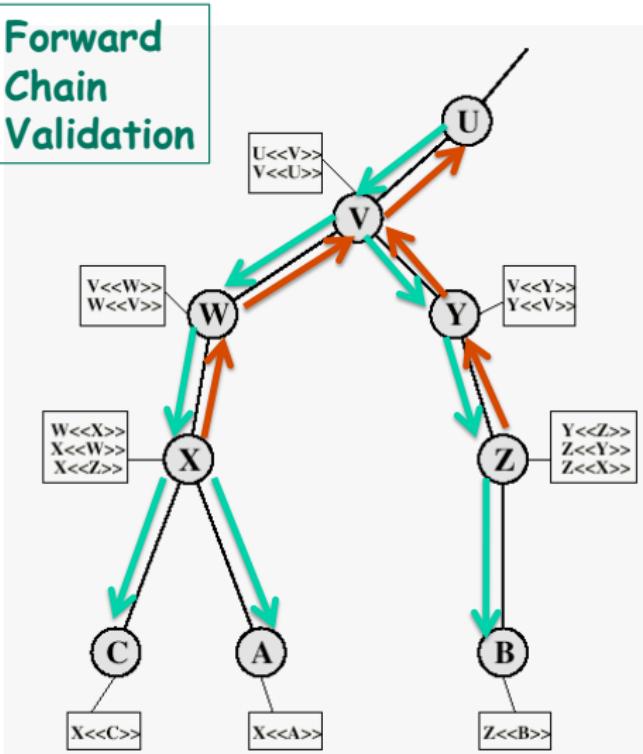
X.509 CA Hierarchy

- Forward certificates
 - Ex., certificates of X, generated by other CAs
- Reverse certificates
 - Ex., certificate generated by X, that are certificates of other CAs

**Forward
Chain
Validation**

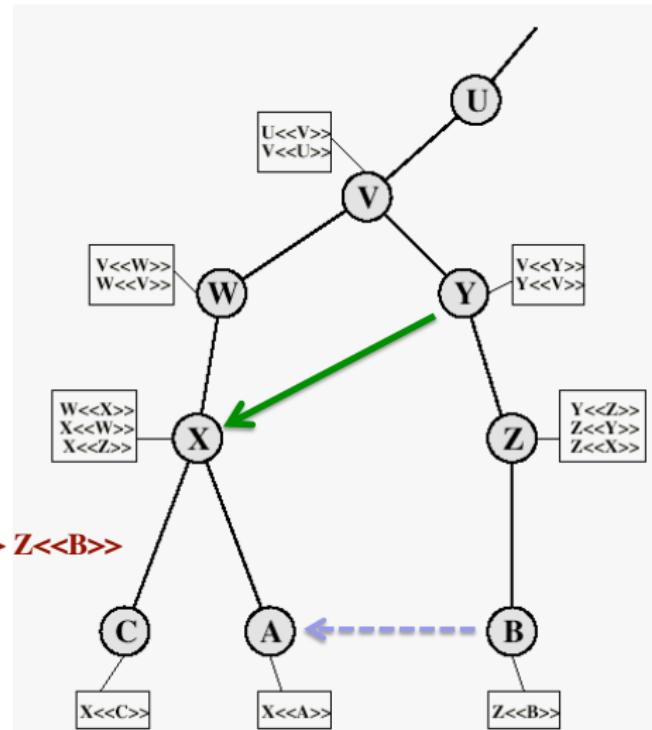
**Forward
Chain
Validation**

**Reverse
Chain
Validation**



X.509 CA Hierarchy

- If A wants to establish a certification path with B with $Z<>$, obtains:



$X<<W>>$ $W<<V>>$ $V<<Y>>$ $Y<<Z>>$ $Z<>$

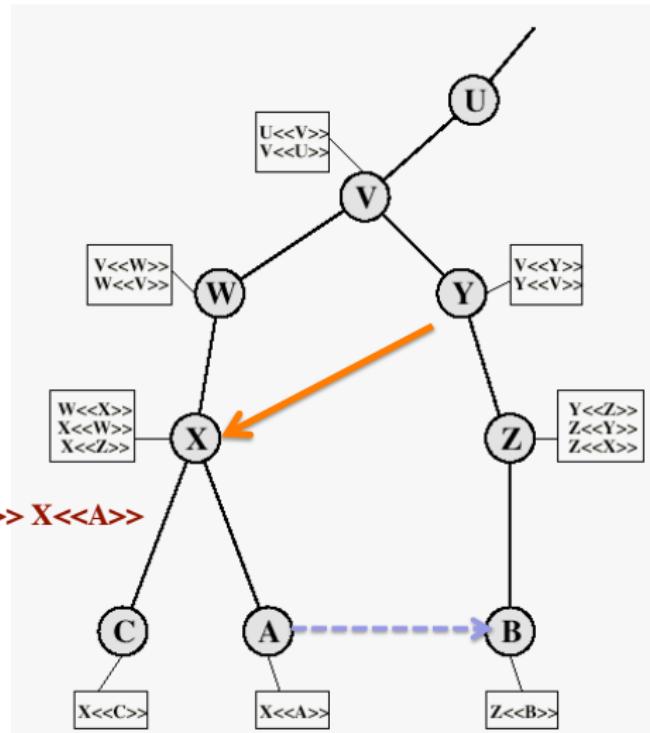
$X<<Y>>$ $Y<<Z>>$ $Z<>$

X.509 CA Hierarchy

- If B wants to establish a certification path with $X<<A>>$, obtains:

$Z<<Y>>$ $Y<<V>>$ $V<<W>>$ $W<<X>>$ $X<<A>>$

$Y<<Z>>$ $Y<<X>>$ $X<<A>>$



Outline

- 
- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate Revocation
 - **PKI - Public Key Infrastructure**
 - PKI Standardization and PKIX Management

Authentication Procedures

One-way authentication and Key dist.

A[$\{ta, ra, Id_B\}K_{ab}$, signData, $\{K_{ab}\}K_{pubB}$]



Two-way (mutual) authentication and Key dist.

A[$\{ta, ra, Id_B\}K_{ab}$, signData, $\{K_{ab}\}K_{pubB}$]

B[$\{tb, rb, Id_A\}K_{ba}$, signData, $\{K_{ba}\}K_{pubA}$]



Three-way (Mutual) authentication And Key Dist.

A[$\{ta, ra, Id_B\}K_{ab}$, signData, $\{K_{ab}\}K_{pubB}$]

B[$\{tb, rb, Id_A\}K_{ba}$, signData, $\{K_{ba}\}K_{pubA}$]

A{rb}



One-Way Authentication

- 1st message (A->B) used to establish
 - the authenticated identity of A and that message is from A
 - message was intended for B
 - integrity & originality of message
- message must include timestamp, nonce, B's identity and is signed by A
- may include additional info for B
 - eg session key, for implicit key-establishment (session key-envelope)
 - Allows the concatenation of additional confidential content or messaging

Two-Way Authentication

- 2 messages ($A \rightarrow B$, $B \rightarrow A$) which also establishes in addition to "one-way":
 - the identity of B and that reply is from B
 - that reply is intended for A
 - integrity & originality of reply
- reply includes original nonce from A, also timestamp and nonce from B
- may include additional info for A
 - May establish "half-duplex" session symmetric keys
 - May establish "full-duplex" session symmetric keys (generated from pre-master keys or exchanged seed-material)

Three-Way Authentication

- 3 messages ($A \rightarrow B$, $B \rightarrow A$, $A \rightarrow B$), adding a final round to mutual authentication
 - which enables above authentication **without dependency from synchronized clocks**
- has reply from A back to B containing signed copy of nonce from B
 - means that timestamps need not be checked or relied upon, preserving anyway message freshness and ordering (protocol termination) control

Authentication Procedures (usage)

Autenticação one-way

Ex., One-Way SSL Authentication, S/MIME or PGP Message Authentication

Autenticação two-way (mútua)

Ex., Two-Way SSL Authentication, SET Protocol

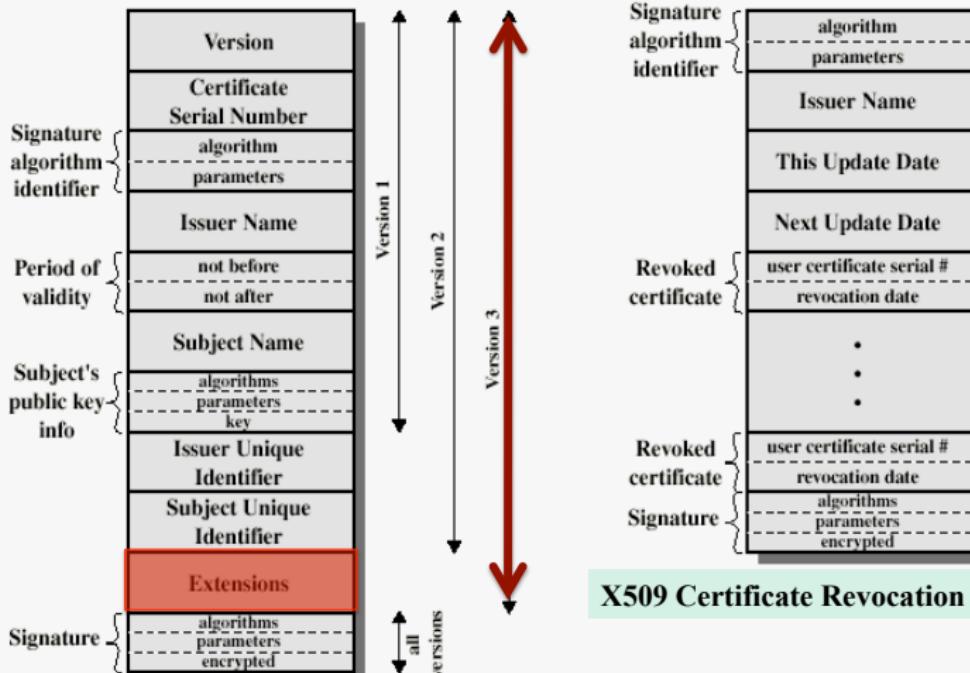
Autenticação three-way (mútua)

Ex., Two-Way SSL Authentication and Key-Session Generation and Agreement

Outline

- 
- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate Revocation
 - **PKI - Public Key Infrastructure**
 - PKI Standardization and PKIX Management

X.509 Certificate and CRL Formats



X.509 certificate (fields in different versions)

X.509 Certificate Revocation List

X509v3

Try / Interpretation of completed certificates and what is involved in complete validation requirements and processes)

- Subject Name (fields and attributes)
 - Not only abstract UIDs, URIs, URLs, eMail addresses, ...
 - Extended with X500 distinguished name attributes and classification categories as well as alternative names
- Issuer name
 - Issuer/CA Distinguished names with X500 attributes
- Certif. policies, policy mapings and key policies
 - Allowing for specific validation to a given policy
 - Setting constraints for limitation/contention of the damage from faulty or malicious CAs
- Inclusion of KeyIDs for Subject and Authority, as Key Selectors
- Information on CRL distribution points, OnLine Status verification points, CA issuers
- Gradual adoption of OID standardization

Outline

- 
- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate Revocation
 - **PKI - Public Key Infrastructure**
 - PKI Standardization and PKIX Management

Extended validation (EV) Certificates

- Introduced by the CA/Browser forum
 - <http://www.cabforum.org/>, http://en.wikipedia.org/wiki/Extended_Validation_Certificate
 - CAs + Relying Party Application Software Suppliers
- Objective: inclusion of standardized procedures for verifying and expressing awareness of the certificate holder and validity (initially motivated by SSL certificates)
- Additional layer of protection: promotion of good practice, guidelines, accurate verification processes for issuing X509v3 SSL certificates
 - Verifying the legal, physical and operational existence of the entity
 - Verifying that the identity of the entity matches official records
 - Verifying that the entity has exclusive right to use the domain specified in the EV Certificate
 - Verifying that the entity has properly authorized the issuance of the EV Certificate

Outline

- 
- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate Revocation
 - **PKI - Public Key Infrastructure**
 - PKI Standardization and PKIX management

Revocation of Certificates

- Certificates are not validated
 - After the expiration
 - Require the issuing of a new certificate just before the expiration of the old one
 - The new certificate can be issued by a different CA
 - If the end use is not according with the content (policies, information extensions)
 - If it is in a certification revocation list (CRL) issued by the CA that issued the certificate
- Reasons for revocation:
 - The user's private key is assumed to be compromised.
 - The user is no longer certified by this CA.
 - The CA's certificate is assumed to be compromised.
 - CA's private keys compromised

Management of CRLs

- Maintained by each CA (or CRL delegation end-points)
 - As a list of revoked but not expired certificates issued by that CA, including
 - End-user certificates
 - Reverse certificates
- Managed by the final users (end-user responsibility)
 - Checked from a directory, every time a certificate is received
 - Supported by OnLine Revocation Protocol
 - CRL endpoint implementing the OCSP protocol
 - Checked from a local cache, periodically updated (ex., BL, Incremental, Time-Controlled)
 - White Lists: White CRLs
 - Black Lists: CRLs
 - Full-Lists vs. Incremental Lists
 - Time-controlled vs. Version-Controlled

Outline

- **X509 Authentication**
 - X509 Standardization
 - X509 Certificates
 - Forward and reverse certification chains
 - Authentication procedures with X509
 - X509 v3 Extensions
 - X509 Extended Validation
 - Certificate Revocation
- **PKI - Public Key Infrastructure**
 - PKIX Standardization and management

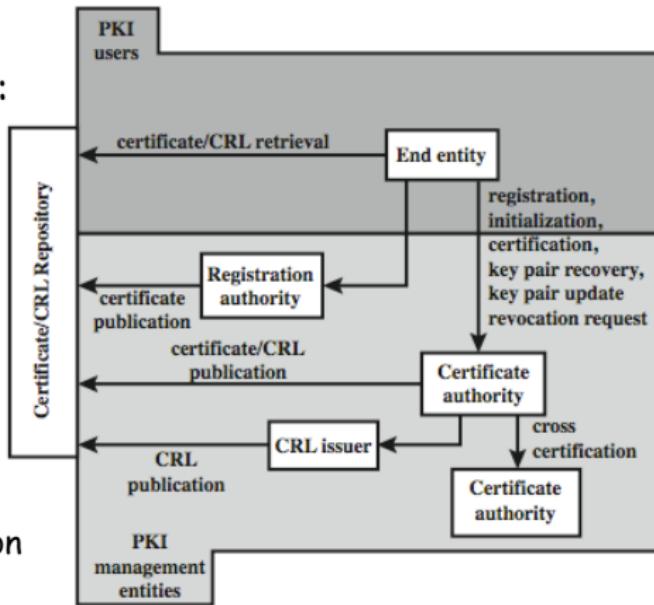


Public Key Infrastructure

- Defined by the RFC 2822
- PKI is the set of: HW, SW, People, Rules, Procedures, Policies and Protocols, needed to create, manage, store, distribute and revoke digital certificates
- Objective: to enable secure, convenient and efficient acquisition of public keys, promoting strict and well-known specifications
- Coordination from the IETF X509 (PKIX) WG
- Standardization base for compatibility purposes on the above issues

PKIX Architectural model and framework

- Key Elements
- Management Functions (APIs):
 - Registration
 - Initialization
 - Certification
 - Key-Recovering
 - Key-Update
 - Revocation Request
 - Cross Certification
- Management Protocols



PKIX Management Functions

- Registration
 - Enrollments from users to CAs (directly or through RAs)
 - Offline and Online procedures for mutual authentication
- Initialization
 - Initialization and installation of trusted CA certificates
- Certification
 - Registration of CSRs to obtain CA issued Certificates in standard formats (ex., PKCS#12, PEM, DER, BASE 64)
- Key Pair Recovery
 - Restoring encryption/decryption keys
- Key Pair Update
 - Regular updates and issuing of new certificates
- Revocation request
 - Regular updates and issuing of new certificates
- Cross certification
 - Exchanged signed CA public keys, between CAs

PKIX Management Protocols

- Standard protocols between PKIX entities supporting PKIX management functions

Ex:

- X509 Internet Public Key Infrastructure - Online certification status protocol: RFC 2560
- CMP - Certificate Management Protocol: RFC 2510
- CMC - Certificate Management Messages over CMS: RFC 2797
- CMS - Cryptographic Message Syntax: RFC 2630

See the standardization process from the
X509 PKIX IETF WG, ... as time goes by ☺
<http://datatracker.ietf.org/wg/pkix/>

Programming support: ex., JAVA PKI API

<http://docs.oracle.com/javase/6/docs/technotes/guides/security/certpath/CertPathProgGuide.html>

X.509 certificate

```

Certificate:
Data:
Version: 3 (0x2)
Serial Number:
        10:ed:fc:82:b7:41:8a:d9:00:5e:45:b6
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=BE, O=GlobalSign nv-sa, OU=GlobalSign Organization Validation CA - SHA256 - G2
Validity
Not Before: Nov 21 08:00:00 2016 GMT
Not After : Nov 22 07:59:59 2017 GMT
Subject: C=US, ST=California, L=San Francisco, O=Wikimedia Foundation, Inc., CN=.wikipedia.org
Subject Public Key Info
    Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
        pub:
            04:c9:22:69:31:ba:d6:6c:ea:da:c3:7f:2c:ac:a5:
                af:c0:02:ea:81:cb:65:bb:fd:0c:6d:46:5b:c9:1e:
                ed:b2:ac:2a:1b:4a:ec:90:7b:e7:1a:51:e0:df:f7:
                c7:4a:20:7b:91:4b:20:07:21:cc:cf:68:65:8c:c6:
                9d:3b:ef:fd:c1
        ASN1 OID: prime256v1
        NIST CURVE: P-256
X509v3 extensions:
    X509v3 Key Usage: critical
        Digital Signature, Key Agreement
    Authority Information Access:
        CA Issuers - URI:http://secure.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt
        OCSP - URI:http://ocsp.globalsign.com/gsorganizationvalsha2g2
X509v3 Certificate Policies:
    Policy: 1.3.6.1.4.1.4146.1.20
        CPS: https://www.globalsign.com/repository/
    Policy: 2.23.140.1.2
X509v3 Basic Constraints:
    CA:FALSE
X509v3 CRL Distribution Points:
    Full Name:
        URL:http://crl.globalsign.com/gs/gsorganizationvalsha2g2.crl
X509v3 Subject Alternative Names:
    DNS:*.wikibooks.org, DNS:*.mediawiki.org, DNS:*.wikidata.org, DNS:*.wikimediawiki.org, DNS:*.wikinews.org, DNS:*.wikipedia.org, DNS:*.wikisource.org, DNS:*.wikiversity.org, DNS:*.wikivoyage.org, DNS:*.wiktionary.org, DNS:*.mediawiki.org, DNS:*.planet.wikimedia.org, DNS:*.wikibooks.org, DNS:*.wikidata.org, DNS:*.wikimedia.org, DNS:*.wikimedialibrary.org, DNS:*.wikinews.org, DNS:*.wikisource.org, DNS:*.wikiversity.org, DNS:*.wikivoyage.org, DNS:*.wiktionary.org, DNS:*.zero.wikipedia.org, DNS:*.mediawiki.org, DNS:*.wiki, DNS:*.wikibooks.org, DNS:*.wikidata.org, DNS:*.wikimedialibrary.org, DNS:*.wikisource.org, DNS:*.wikiversity.org, DNS:*.wikivoyage.org, DNS:*.wiktionary.org, DNS:*.wikifusioncontent.org, DNS:*.wikimedia.org
X509v3 Extended Key Usage:
    TLS Web Server Authentication, TLS Web Client Authentication
X509v3 Subject Key Identifier:
    2B:2A:26:2A:57:BB:3B:CE:B4:06:AB:54:E7:D7:3B:21:C1:49:5C:36
X509v3 Authority Key Identifier:
    keyid:96:DE:61:F1:BD:1C:16:29:53:1C:CC:7D:3B:83:00:40:E6:1A:7C
Signature Algorithm: sha256WithRSAEncryption
8b:c3:ed:d1:9d:39:6f:af:40:72:bd:1e:18:5e:30:54:23:35
...

```

Key Exchange

Part I

- Introduction
 - What is a Key Exchange Protocol
 - Formalization and Design Challenges
- Main examples: Authenticated Diffie-Hellman
- Common pitfalls and sound design principles
 - Learning by examples and counter-examples
- Formalizing and proving key exchange protocols
 - Modular design and analysis: Authenticators
- Protocols
 - ISO and SKEME
 - STS and Photuris
 - SIGMA Protocol and identity privacy
- IPsec's Internet Key Exchange (IKE)

Part II

- Implicit authenticated key exchange
 - In the complex search for extreme simplicity
 - HMQV, Okamoto-Tanaka, One-Pass KE

Part III

- Key derivation & the HKDF extract-and-expand scheme

What is a Key Exchange Protocol

- Many answers: From naïve intuition to rigorous formal theory
- The fundamental role of KE
 - Bootstrap a secure channel between two communicating parties via the negotiation of shared keys and cryptographic services
 - Enabler and **heart** of secure communications
 - The link between long-term keys and fresh session keys
 - Link between public-key and symmetric cryptography
- **The most common form of cryptographic protocol in wide use**
 - Can teach us a lot about design and analysis of more complex protocols
 - **Deceptively simple** (overlooked as a "given" in multi-party protocols)

Key Exchange Protocols

(very informal)

- A protocol between two parties to establish a shared key (“session key”) such that:
 1. **Authenticity**: they both know who the other party is
 2. **Secrecy**: only they know the resultant shared key
- Also crucial (yet easy to overlook):
- 3. **Consistency**: if two honest parties establish a common session key then both have a **consistent view of who** the peers to the session are
- A: (B,K) and B: (id,K) \rightarrow id=A

Key Exchange Protocols

- More generally:
 - Multiple parties; any two may exchange a key
 - Sessions: multiple simultaneous executions
- Adversary:
 - Monitors/controls/modifies traffic (man-in-the-middle)
 - May corrupt parties: learns long-term secrets
 - May learn session-specific information: state/keys
- Security goal: preserve authenticity, secrecy and consistency of uncorrupted sessions
 - Confine damage from exposure to a minimum

Formalizing Key Exchange

- An intuitive notion but hard to formalize
- Wish list for formal definitions/model:
 - Intuitive (beware!)
 - Reject “bad” protocols (capture full capabilities of realistic attackers)
 - Accept “good”, natural protocols (avoid overkill requirements)
 - Ensure security of KE applications: “secure channels” as the quintessential application (protocol composition)
 - **Usability:** easy to analyze (stand alone → composable)
+ a design tool

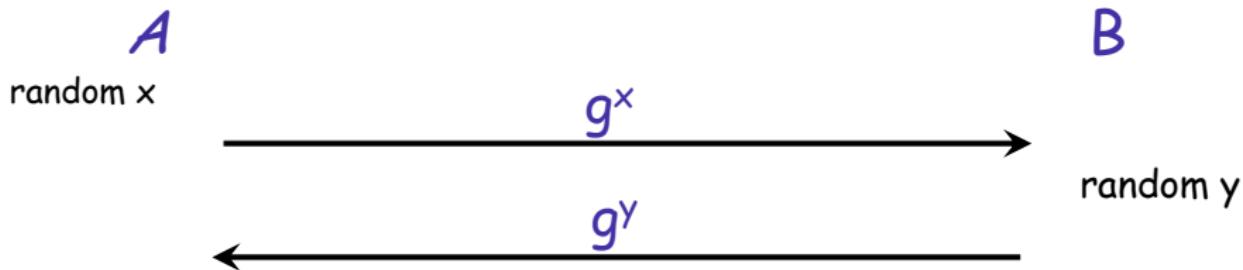
Designing and Analyzing KE Protocols...

- ...is non-trivial
- Yet the end protocol need not be complex
 - And: to be practical the protocol *MUST BE SIMPLE*
- Best advice: learn from past experience (good and bad)
- Formal analysis as an indispensable tool
 - But remember: there is no ULTIMATE security model or absolute proofs of security
 - only relative to the model (and cryptographic assumptions)

Diffie-Hellman and Key Exchange Protocols

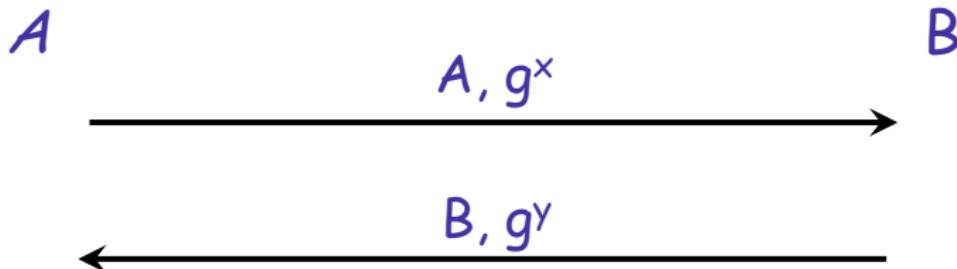
Diffie-Hellman Exchange [DH'76]

g = generator of a cyclic group G (e.g. \mathbb{Z}_p^* , EC group)



- Both parties compute the secret key $K = g^{xy} = (g^x)^y = (g^y)^x$
 1. CDH Assumption: K hard to compute given only g^x and g^y
 2. DDH Assumption: K indistinguishable from random element in G
 - Decisional DH assumption: $(g^x, g^y, g^{xy}) \approx_c (g^x, g^y, g^{\text{random}})$
- From g^{xy} to a k-bit key: KDF: $g^{xy} \rightarrow \{0,1\}^k$ (pseudorandom)

Diffie-Hellman KE and PFS

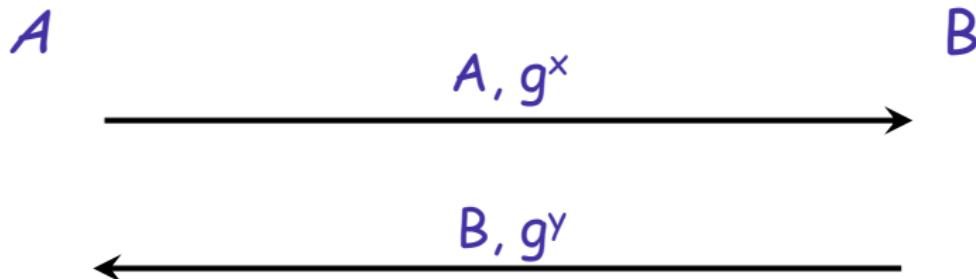


■ Perfect Forward Secrecy (PFS)

- Once the session keys are destroyed there is no way to recover them, not even by the owners (not even at gunpoint)
- Distinguishes D-H from other protocols
 - compare SSL: What if your bank's private encryption key ever compromised? ALL past traffic exposed!
 - With PFS long-term keys used only for authentication

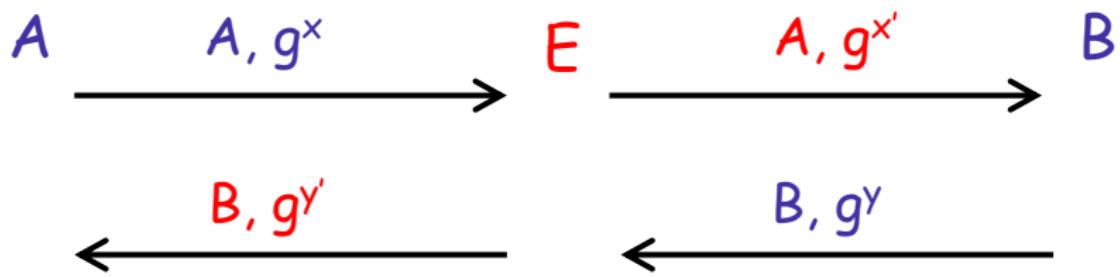


Diffie-Hellman Exchange [DH'76]



- beautiful, strong, but...
- secure only against eavesdroppers
- open to active attacker (man-in-the-middle)

(Wo)Man-in-the-Middle



$$K_{AB} = g^{xy'}$$

$$K_{BA} = g^{x'y}$$

Eve knows both keys!

The Long Journey Towards Authenticated DH Protocols

- UN-authenticated DH has survived to this date without change (40+ years!)
- In the same period, hundreds of papers published on *authenticated DH protocols, many (most?) of them broken!*
- Why is it so hard to get it right?

The Long Journey Towards Authenticated DH Protocols

- Why is it so hard?
 - What is authentication? Difficult notion, non-trivial to formalize.
Changes with trust and setup scenarios.
 - How do we know when a protocol is secure?
When are we done debugging it? Poor track record...
- Took long time to be able to develop sound, general security models
- And even more to prove protocols in these models
- And guess what... unproven protocols tend to be broken

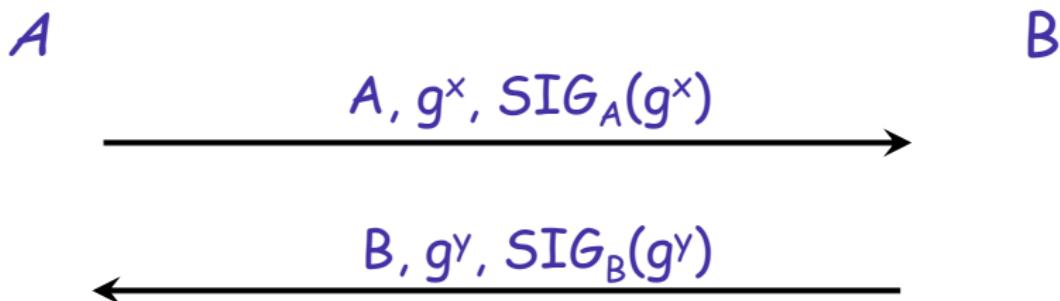


Guided Tour to Authenticated DH Protocols

Conventions (and disclaimers)

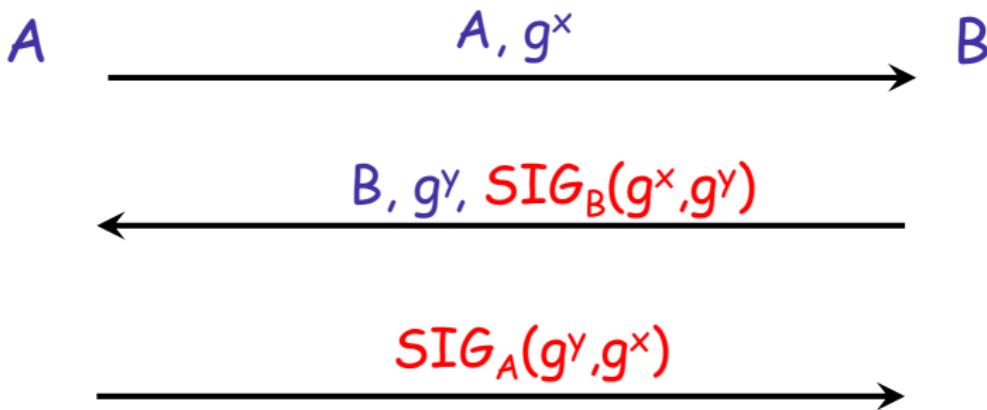
- A, B denote participants as well as their identities.
- Certificates: We assume parties have long-term public keys which other parties can learn and verify
 - Either by out-of-band verification or via certificates sent in the protocol (in the latter case A, B stand for identities plus certificates)
 - We omit many essential operations such as explicit verification operations (of signatures and certificates)
- DH group: We use "universal parameters" (e.g. p, g) - these can be wired into the protocol, negotiated during execution, etc.
- Note: No protocol can be secure without careful treatment of these "details" (remember: in crypto, the devil is in the details)
- Here the focus is on basic structure and general principles

First Attempt at Authenticated DH



- what if attacker ever finds a triple $(x, g^x, SIG_A(g^x))$?
 - E.g., file of precomputed (x, g^x) pairs
- Violates basic principle: Ephemeral leakage should not allow for long-term impersonation (beyond the leaked session)

Basic Authenticated DH ("BADH")



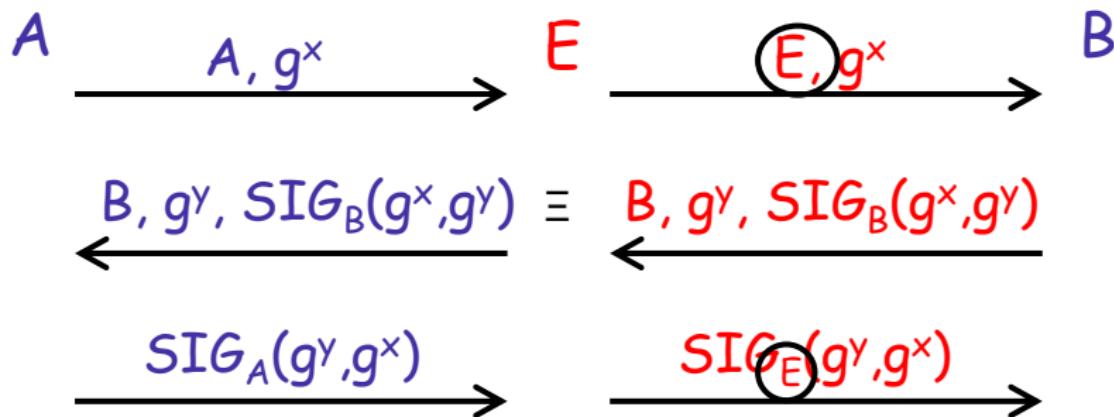
Each party signs its own DH value to prevent m-i-t-m attack
and the peer's DH value as a freshness guarantee against replay

A: "Shared $K=g^{xy}$ with B" ($K \leftrightarrow B$) B: "Shared $K=g^{xy}$ with A" ($K \leftrightarrow A$)

Looks fine, but...

Identity-Misbinding Attack [DVW'92]

(a.k.a. Unknown Key-Share attack = UKS)



- Any damage? Wrong identity binding!

A: "Shared K= g^{xy} with B" ($K \Leftrightarrow B$) B: "Shared K= g^{xy} with E" ($K \Leftrightarrow E$)

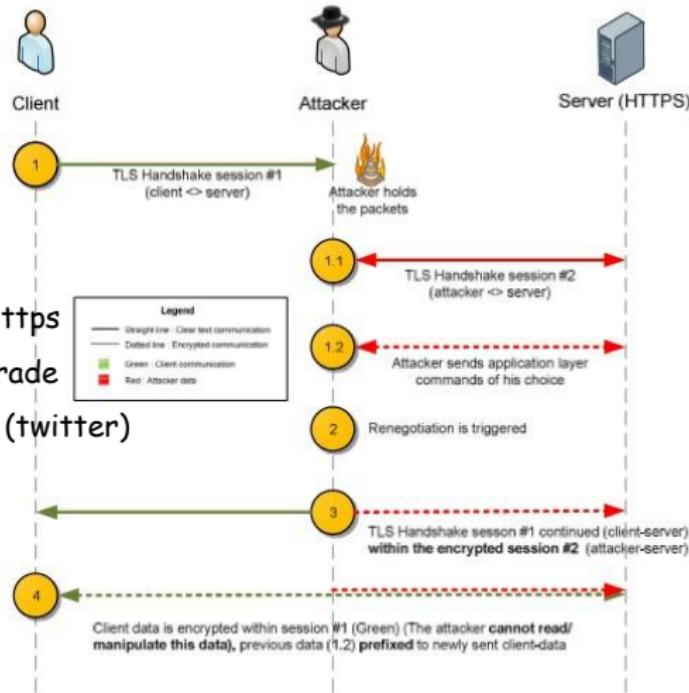
E doesn't know K= g^{xy} but B considers anything sent by A as coming from E

A: "Shared $K=g^{xy}$ with B" ($K \leftrightarrow B$)

B: "Shared $K=g^{xy}$ with E" ($K \leftrightarrow E$)

- $B = \text{Bank}$ $A, E = \text{customers}$
 - $A \longrightarrow B: \{\overset{\text{post this page on my website}}{\cancel{\text{"deposit \$1000 in my account"}}}\}_K$
 - $B \overset{\text{posts the page}}{\longrightarrow} \overset{\text{website}}{E}$
 - B deposits the money in " K "'s account, i.e. E 's!
-
- Should the bank protocol include explicit identities? Maybe, but KE should not make assumptions on higher-layer mechanisms
 - What is the expectation of higher layer protocols? That a key is uniquely bound to its owners ("speaks for its owners")
 - SSL renegotiation's bug: wrong binding of sessions (attack succeeded without the attacker ever learning the key)

4. Generic TLS renegotiation prefix injection vulnerability

Some exploits:

- Command injection https
- https to http downgrade
- Stealing credentials (twitter)

Yet another example

HOME PAGE | TODAY'S PAPER | VIDEO | MOST POPULAR | U.S. Edition ▾

The New York Times Middle East

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS

AFRICA | AMERICAS | ASIA PACIFIC | EUROPE | MIDDLE EAST



RATIONAL
MIDDLE
ENERGY SERIES

LET'S CREATE
TOWARDS OUR
JOIN THE CONVERSATION

Iranians Say They Took Secret Data From Drone

By STEVEN LEE MYERS and SCOTT SHANE
Published: April 22, 2012

WASHINGTON — Iran claimed on Sunday to have extracted secret intelligence information from an advanced American drone aircraft that crashed in the country in December, seeking a propaganda victory over what has been an embarrassing intelligence failure for the Central Intelligence Agency.

The air force commander of Iran's



F FACEBOOK | T TWITTER | G GOOGLE | E E-MAIL | S SHARE | P PRINT

Drone Example:

A: "Shared $K=g^{xy}$ with B" ($K \Leftrightarrow B$)

B: "Shared $K=g^{xy}$ with E" ($K \Leftrightarrow E$)

- A= F-16 B= Central Command E= drone (in enemy hands)
- B → E: {"destroy yourself"}_K
- E passes command {"destroy yourself"}_K to A.
- Result: F-16 destroys itself!

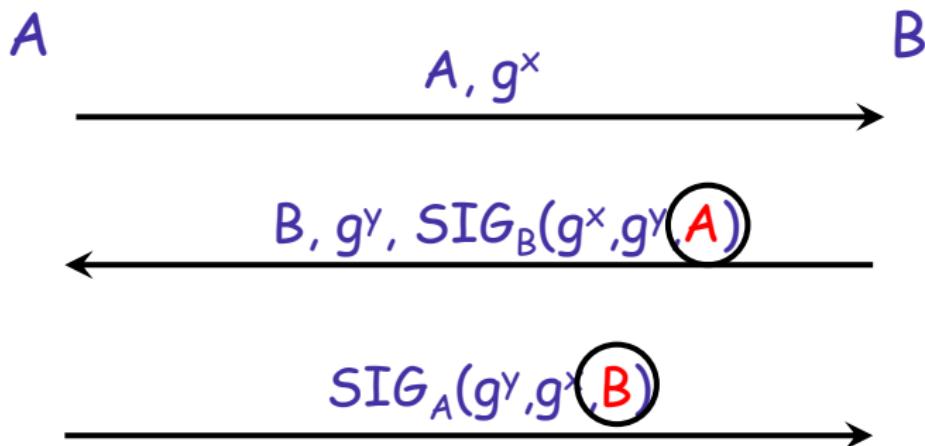
Notes

- Attack discovered by Diffie-van Oorschot-Wiener [DVW'92]
 - the “differential cryptanalysis” of KE protocols
 - highlights the crucial consistency property

$A: (B, K)$ and $B: (id, K) \rightarrow id = A$

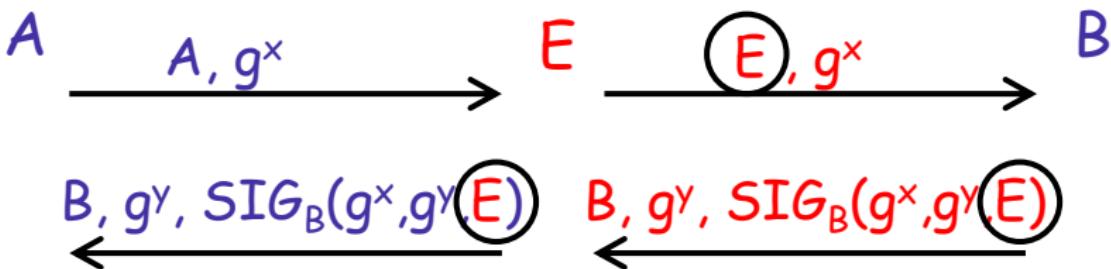
- Note: The terminology Identity Misbinding Attack is mine
The attack is more commonly referred to as the Unknown Key-Share (UKS) attack.

A Possible Solution (ISO-9796)



Thwarts the identity-misbinding attack by including the identity of the peer under the signature

The ISO defense



A: aha! B is talking to E not to me!

Note that E cannot produce $\text{SIG}_B(g^x, g^y, A)$

- The ISO protocol thus avoids the previous mentioned attacks; **but is it secure??**

The ISO Protocol is Secure

- We'll sketch the proof in the SK-security model of key exchange (known as the Canetti-Krawczyk model [CK'01])
- Note: the actual ISO-9796 protocol is more complicated: adds a MAC on the peers id -- which adds nothing to the security of the protocol
- An important consequence of well-analyzed protocols:
avoiding “safety margins”
 - PROOF-DRIVEN DESIGN®

Let's then talk about KE models and proofs...

On KE Analysis Work

- Two main methodologies
 - Complexity based: security against computationally bounded attackers, proofs of security, reduction to underlying cryptography, probabilistic in nature
 - Logic-based analysis: abstracts crypto as ideal functions, protocols as state machines, good protocol debuggers
- Recent bridging work towards “the best of two worlds”
 - The power of automated analysis shown in recent TLS 1.3 design
- Here we focus on the first approach

Remember a “definition desiderata”

- Intuitive (session notion, attacker capabilities, secrecy)
- Reject “bad” protocols (capture full capabilities of realistic attackers)
- Accept “good”, natural protocols (avoid overkill reqt's)
- Ensure security of KE applications: “secure channels” as the quintessential application + composition
- Usability: easy to analyze (stand alone → composable)
+ a design tool
- One more confidence source: Equivalence of different definitions

From Turing on Definitions of Computability

- A. Turing. On Computable Numbers, With an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230-254, 1937.
- No attempt has yet been made to show that the "computable" numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. The real question at issue is "What are the possible processes which can be carried out in computing a number?"
- The arguments which I shall use are of three kinds.
 - a) A direct appeal to intuition.
 - b) A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal).
 - c) Giving examples of large classes of numbers which are computable.

CK Model Predecessors

- Bellare-Rogaway'93
 - First complexity-theoretic treatment of KE
 - Indistinguishability approach [GM84]: attacker can't distinguish the real key from a random one
 - Extended in [BJM97] to the PK-authentication setting
- Bellare-Canetti-Krawczyk'98
 - Simulation-based definition of KE security
 - Ideally-authenticated (AM) vs. real-life (UM)
 - Modular authentication methodology
 - **Authenticators: AM-to-UM compilers**
- Both works required tunings (learning is a never-ending process)

Canetti-Krawczyk Model

- A combination of BCK'98 setting (simulation) and BR'93 indistinguishability approach (“SK-security”)
 - Goal: ensure good composition and modularity properties but keep the simplicity of indistinguishability-based analysis (“usability”)
 - Later years: BR - CK convergence
- Secure channels as the “test application”
 - Requires a formalization of secure channels (e.g., a transport protocol such as IPSec, SSL, SSH) - simplifies, see KP talk
- Universally Composable security

duplicate

SK Security [CK'01]

- Geared towards allowing protocol composition [BCK'98] especially with generic "secure channels protocol"
- Follows indistinguishability approach [BR'93]
 - Defines a set of possible adversarial interventions/corruptions
 - Requires that such an attacker **cannot distinguish the shared session key from random** (as long as the parties to the session and the session itself are not corrupted)
- Secure channels as the must "test application"
 - Requires a formalization of secure channels (e.g., a transport protocol such as IPSec, SSL, SSH) - more later

SK-Security: KE protocol

- A two-party protocol in a multi-party setting
- Multiple protocol executions may run concurrently at the same or different parties
- Each run of the protocol at a party is called a session (a *local object*)
- Sessions have a unique local name: e.g. (A, s_A) and an incoming name (B, s_B) where B is the intended *peer*.
The session id is the concatenation: (A, s_A, B, s_B)
- Sessions with corresponding names, i.e., (A, s_A, B, s_B) and (B, s_B, A, s_A) are called matching.
- Upon completion a session erases its state and outputs a triple: (session-id, peer-id, session-key)

SK-Security: Attacker

- Adversary model (Unauthenticated links Model - UM)
 - Full control of communication links: monitors/controls/modifies traffic (m-i-t-m)
 - Schedules KE sessions at will (interleaving)
 - May corrupt parties (total control): learns long-term secrets* (e.g. signature key), all its state and session keys
 - May issue a “learning query” for short-term information:
 - session state query (e.g., the exponent x of a g^x value)
 - session key query (of a complete, present or past, session)
- Exposed session: if session owner is corrupted, or attacker issued a query against the session, or the matching session is exposed
 - Clearly cannot protect a session if the matching is exposed

A KE Protocol is called SK secure if

1. Completed matching sessions output same session key
(functional, non-triviality clause)
2. Attacker learns *nothing* about *unexposed* sessions
 - Captured via “test session”; chosen by attacker among completed *unexposed* sessions
 - Attacker is given either the session key or an independent random key; it needs to guess which one is the case
 - Require that the probability that the attacker guesses right is not significantly better than a random guess (i.e. $\frac{1}{2} + \text{small } \epsilon$)

SK-Security Definition (simplified)

A KE protocol is SK-secure (in the above adv. model) if for any session (P,s) that completes at an uncorrupted party P with peer Q it holds:

1. If Q completes session (Q,s) while P and Q are uncorrupted then:
 - a) $\text{peer}(Q,s)=P$; and
 - b) $\text{sk}(Q,s)=\text{sk}(P,s)$
2. If sessions (P,s) and (Q,s) are *not exposed*, attacker cannot distinguish $\text{sk}(P,s)$ from a random value

The def'n allows the attacker to learn the key of session (P,s,Q) by exposing that session (via session key/state query or via party corruption) or by exposing its "matching session" (Q,s,P) . In all other cases it should learn nothing about the SK!

A compact but strong definition

- Captures many attacks that were *enumerated* in the past as separate requirements (or wish lists). For example:
 - Impersonation: if E can impersonate Bob without corrupting him then E knows a key of an unexposed session, contradicting the definition
 - Secrecy: If E learns anything about the session key then it can distinguish it from random. (Note: Why indistinguishability? The OTP example.)
 - Known-key attacks: An important class of attacks studied separately in the past: Can E break one session given the key of another session?
Captured via session key query
 - Identity misbinding: if E forces two sessions w/outputs (A, B, K) and (B, E, K) , then E can choose one as test and expose the other to learn K (it is allowed to do so since sessions are not matching)
- The definition can be further extended to cover other threats and security properties: e.g. PFS (via key expiration)

Informal Summary

- Summary of security guarantees for honest A,B:
 1. If A outputs session key (A, B, K) and B is honest then no one except B may know anything about K (not even a single bit)
 - Does the protocol guarantee that B outputs the key??
 - “key confirmation” possible but “common knowledge” is not
 2. Session keys are “computationally independent” of each other
 - Note: *session keys cannot be used during the key exchange itself* (cf. TLS)
- Note: Sharing a key with a corrupted party:
 - No guarantee for a key shared with a corrupted party.
 - But there is a guarantee that interacting with the attacker does not compromise any session between honest parties

Informal Summary

- Summary of security guarantees for honest A,B:
 1. If A outputs session key (A, B, K) and B is honest then no one except B may know anything about K (not even a single bit)
 - Does the protocol guarantee that B outputs the key??
 - “key confirmation” possible but “common knowledge” is not
 2. Session keys are “computationally independent” of each other
 - Note: *session keys cannot be used during the key exchange itself* (cf. TLS)
- Note: Sharing a key with a corrupted party:
 - No guarantee for a key shared with a corrupted party.
 - **Beware! Triple Handshake Attack**
But there is a guarantee that interacting with the attacker does not compromise any session between honest parties

About public keys

- Parties have long-term secrets
 - private signature/decryption keys, or shared keys w/other parties
- In the PK case: public keys of honest parties are assumed to be communicated to other parties correctly.
- Public keys of corrupted parties are chosen by the attacker arbitrarily (e.g., may be equal to a public key of another honest party).
 - Think of a CA that checks identity but no other properties of the keys being registered (does not assume/require proof of possession, checking structure of a key, etc.)

SK-security results (secure channels)

- SK-security → Secure Channels authenticated
encryption
- Any key exchanged with an ~~SK~~-secure KE protocol and used to "encrypt-then-authenticate" data realizes a secure channel [CK01]
- Secure channel realization:
 - $K = \text{output of KE}$ (may require a KDF step, e.g. if $K=g^{xy}$; can be combined with prf step below)
 - Keys ($K_{\text{enc}}, K_{\text{mac}}$) = $\text{prf}_K(\text{context})$ where context may include session/protocol/algorithim identifiers, parties identities, etc.
 - Note: having directional keys ($A \rightarrow B$ and $B \rightarrow A$) is a good idea
 - Apply to data: $c = \text{Enc}_{K_{\text{enc}}}(\text{data})$, $t = \text{Mac}_{K_{\text{mac}}}(c)$; transmit (c, t)
- Other combinations, e.g. $\text{Enc}_{K_{\text{enc}}}(\text{data} \parallel \text{Mac}_{K_{\text{mac}}}(\text{data}))$ may not be secure (TLS examples)

SK-security results (protocols)

- A variety of protocols have been proven SK-secure (both DH and key-transport)
 - e.g., ISO, IKE (SKEME, SIGMA), HMQV, Pre-Shared and more
 - Two SK-secure flavors: with and w/o PFS
 - PFS modeled through session-expiration (models erasure); expired sessions are NOT exposed even if attacker corrupts the session's owner.

SK-Security and Composition

- CK02: SK-Security is “universally composable” (UC [Can'02])
 - Remains secure under composition with any application, not just secure channels
 - Well, almost: true for protocols with the ACK property
 - True always if UC security weakened via “non-information oracles” (see CK02 eprint/2002/059)
- SK-Security preserved under *authenticators*
 - Authenticator: A “compiler” from (ideal) AM-secure protocols to (realistic) UM-secure protocols (a design and analysis tool!)
 - More in following slides (incl. the proof of the ISO protocol)

UM and AM Models

AM

- Adversary model: ~~UnAuthenticated~~Authenticated links Model (AM)
 - Full control of communication links: monitors/controls/~~modifies~~ traffic (m-i~~n~~-m) **passive attacker**
 - Schedules KE sessions at will (interleaving)
 - May **corrupt** parties (total control): learns long-term secrets (e.g. signature key), all its state and session keys
 - May issue a “learning query” for short-term information:
 - session state query (e.g., the exponent x of a g^x value)
 - session key query (of a complete, present or past, session)
- **Exposed session:** if session owner is corrupted, or attacker issued a query against the session, or the matching session is exposed
 - Clearly cannot protect a session if the matching is exposed

Authenticators [BCK98]

- Models:
 - UM (Unauthenticated-links Model): a realistic attack model as described before
 - AM (ideally Authenticated-links Model): like UM but attacker is passive; cannot change or inject msgs on links (but it may prevent delivery)
- *Authenticator* : a “compiler” from AM-secure protocols to UM-secure
 - Reduces the problem of designing (and analyzing) protocols from the complex UM to the simple AM
 - For example: Proving plain DH in the AM is immediate (under DDH assumption)

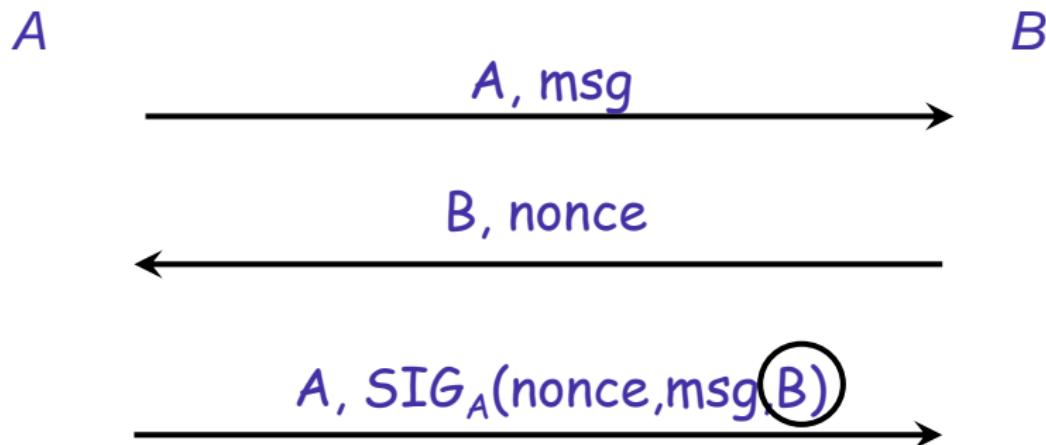
Authenticators (sketch of idea)

- Message sending protocol (can be interactive)
 - Parties send and receive messages and register their actions (“sent msg m to B ”, “received msg m from A ”)
- An *authenticator* is a message sending protocol s. t.
 - Whenever A registers “received m from B ”, it also holds that B registered “sent m to B ”
 - Note: To prevent replay attacks messages need to be made unique (e.g., concatenated with msg id or nonce)

Authenticators Theorem

- Theorem: Let A be an authenticator
 - If P is a protocol secure in the AM model
 - and P' is the result of applying A to each message in P
- Then, P' is secure in the UM model.

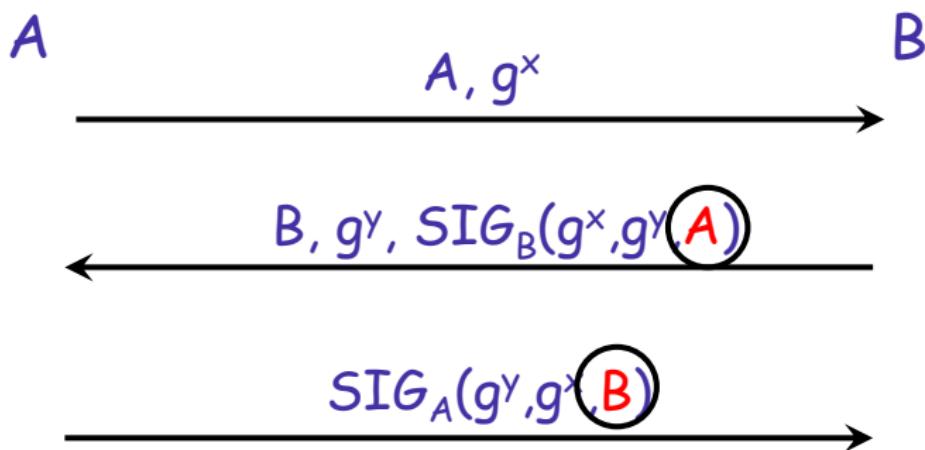
A signature-based authenticator (applied to message msg)



Compiler from AM to UM: apply the above authenticator to each protocol's message

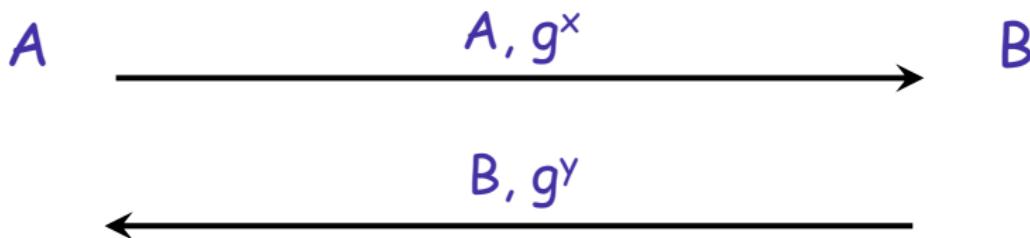
Proving ISO Using an Authenticator

Recall the ISO-9796 protocol
(solved the Identity-Misbinding Attack of BADH):



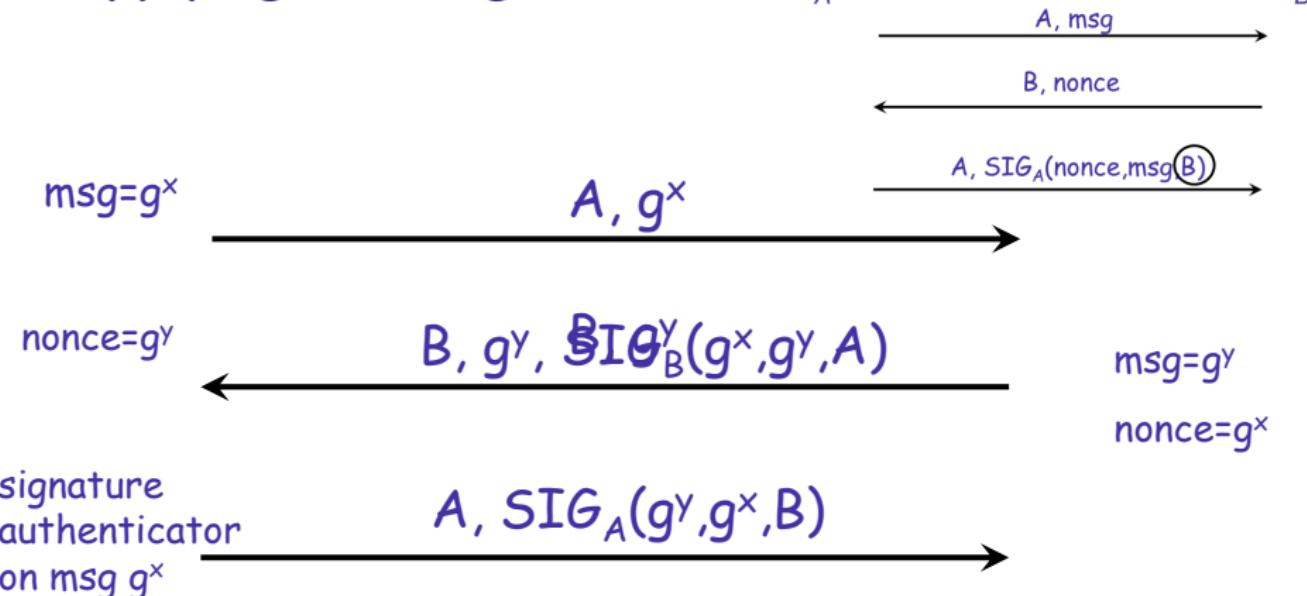
Proving ISO Using an Authenticator

- First prove basic DH is SK-secure in AM
 - Equivalent to Decisional DH assumption: $(g^x, g^y, g^{xy}) \approx_c (g^x, g^y, g^{\text{random}})$
i.e., g^{xy} indistinguishable from random element in G



- Next apply the sig-based authenticator to this protocol → a proof of the ISO protocol!!

Applying the Sig-Authenticator to AM-DH



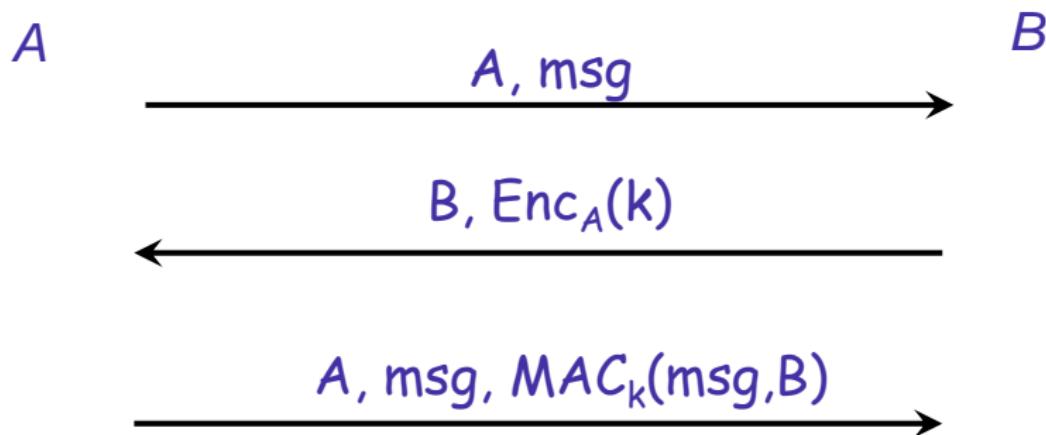
Authenticator is applied to msg g^x is significantly different than authenticator
first A sends nonce (g^x), then B sends message (g^y) with signature

Conclusion: the ISO protocol is SK-secure
(QED: with a simple and intuitive proof)

Other Authenticators (and the SKEME Protocol)

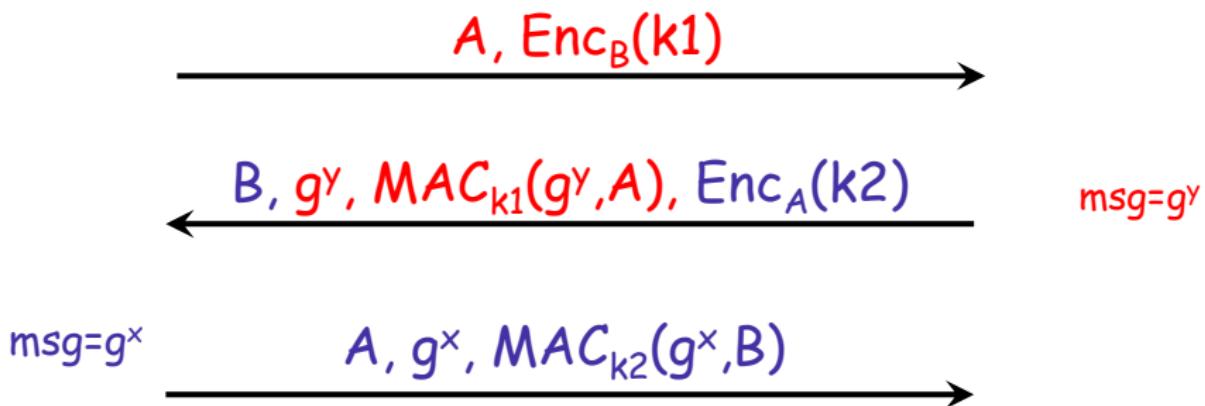
PK-Encryption-based authenticator

Single message authenticator: $A \xrightarrow{\text{msg}} B$:



Compiler from AM to UM: apply the above authenticator to each protocol's message

Applying the Enc-Authenticator to AM-DH



→ the SKEME protocol [K'96, IKEv1]

- Variants:
- Key transport (no pfs)
 - Pre-shared key (with a MAC-based authenticator)

Authenticators are not always...

- Possible
 - Either the design is not decomposable into a basic AM-secure protocol and an authenticator applied to it
- Or desirable
 - The decomposition is artificial and adds more technicalities than understanding
- Yet when they “work” it usually results in a more intuitive, modular and easier-to-analyze protocol
 - And designing KE with authenticators in mind reduces the chances of hidden flaws

More on the Design of Key Exchange Protocols

- Privacy Issues: Identity Protection, deniability
- The design of the IKE Protocols: SKEME, SIGMA
 - “IPsec’s Key Exchange” (IKEv1, IKEv2)

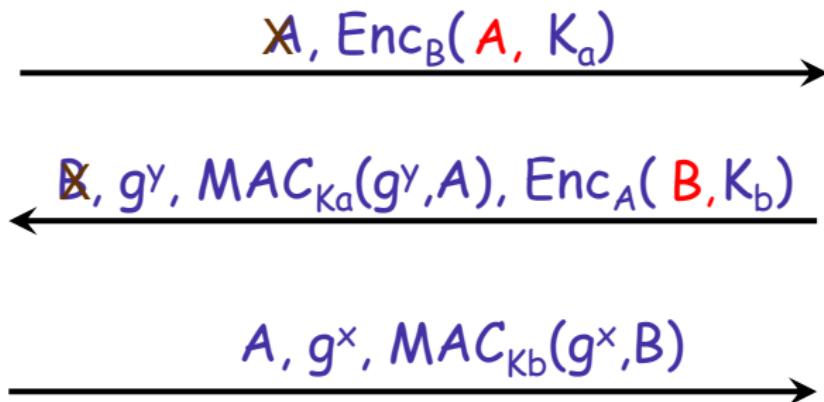
On Identity-Protecting KE Protocols

- Identity protection
 - Hiding identities from passive and/or active attackers
 - Logical identities (e.g. cert's) vs. physical addresses
- A privacy concern in many scenarios
 - Probing attacks in the Internet: who are you?
 - Location anonymity of roaming users
 - The “intelligent passport” application
- IPSec/IKE: design highly influenced by such privacy concerns (→ SKEME, SIGMA)

Identity Protection

- Passive vs. active attacker
 - Both id's protected against passive attacks but only one against active attacks
 - Which identity should get active defense?
 - Initiator: roaming user (e.g. hide location)
 - Responder: avoid probing attacks: who are you? (e.g. passport)
- Presents some design challenges: conflict between anonymity and authentication

Identity Protection in SKEME

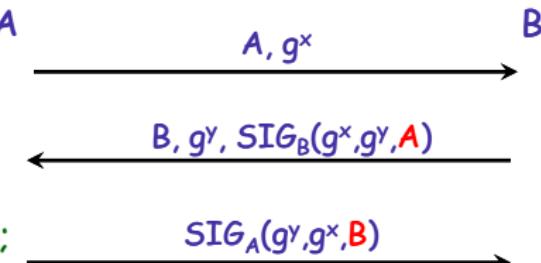


Issue: Id protection requires A to know B's pk (before run)

Next: SIGMA (signature based, solves this issue, IKEv2)

Why not ISO?

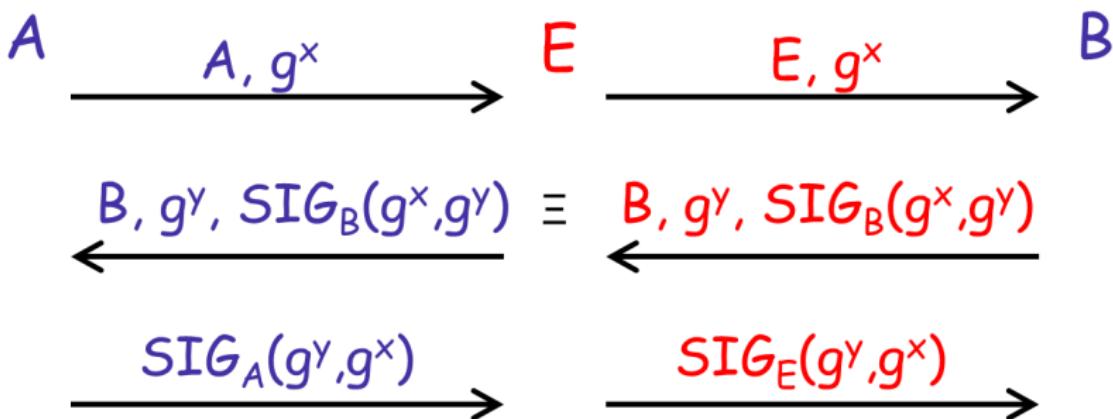
Unsuited for identity protection



- B needs to know A's identity before he can authenticate to A; same for A
 - Protection against active attackers is not possible
- Another privacy concern: leaving a signed proof of communication (signing the peer's identity)
- Letting each party sign its own identity rather than the peer's solves the privacy issues but makes the protocol insecure (the identity-misbinding attack again)

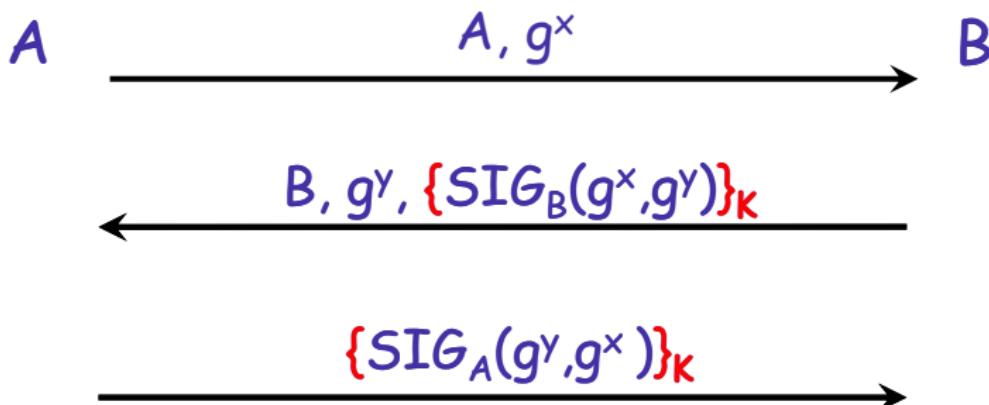
Alternative Solution: STS [DVW'92]

- Idea: to prevent the Id-M attack against BADH,
A and B "prove knowledge" of $K=g^{xy}$ to each other
- Reminder Id-M attack (note that E doesn't know g^{xy})



Alternative Solution: STS [DVW'92]

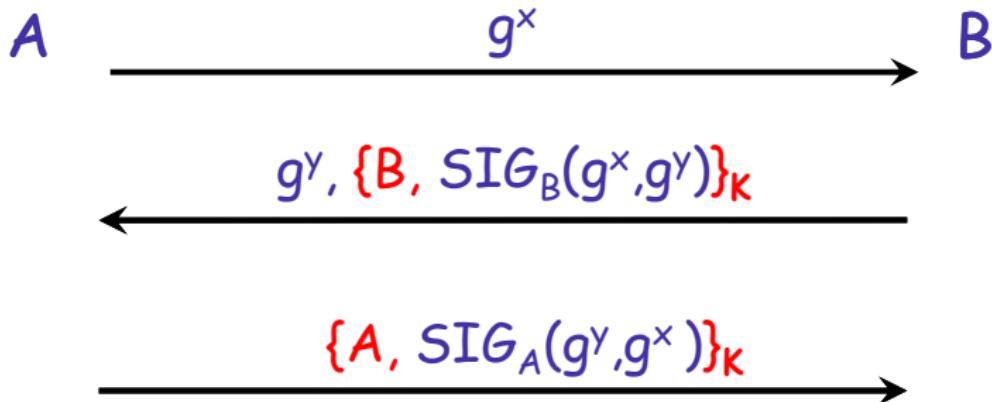
- Idea: each peer proves knowledge of $K=g^{xy}$
(prevents the Id-M attack since in BADH E doesn't know g^{xy})
- As a "Proof of Knowledge" the STS protocol uses encryption under $K=g^{xy}$ (encryption denoted by $\{\dots\}_K$)



STS Pro's and Con's

😊 Pro: STS can protect identities

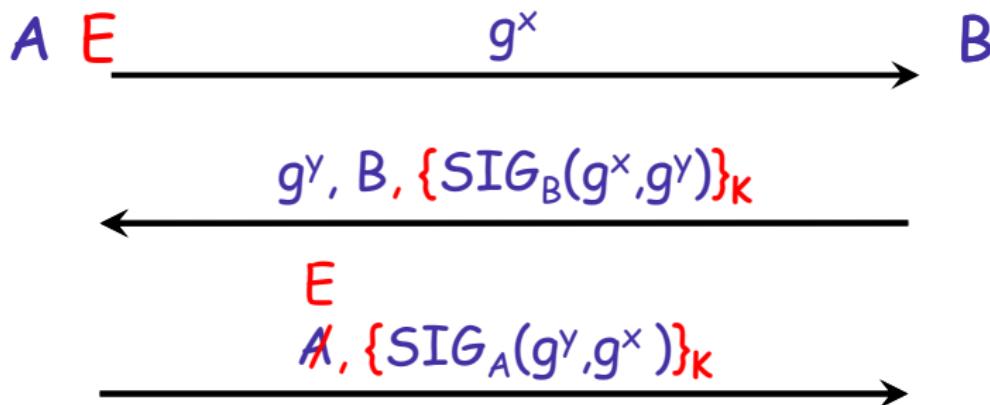
- Peer's id not needed for your own authentication
- Can extend encryption to cover identities (or cert's)



STS Pro's and Con's

:(:(Con: Protocol is insecure! (encryption is not the right function to prove knowledge of a key)

- E.g.: if Eve can register A's public-key under her name she can mount the I-M attack (without knowing g^{xy})

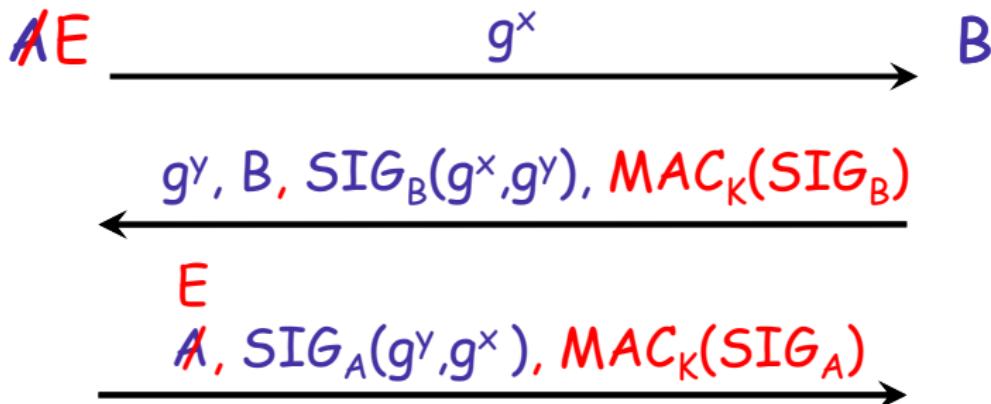


Identity-Misbinding on STS

- Eve registers A's PK as her own PK
 - Applicable when CA checks for identity of registrant but not for "possession" (PoP) of private key
- The attack is trivial if cert's not encrypted and trivial too if encrypted with a stream cipher
- Beyond the practicality of the attack, it is enough to show that "proof of knowledge of g^{xy} " via encryption is not enough.

MOREOVER...

STS with MAC (instead of encryption) [DVW]



- MAC_K better suited to provide Proof of Knowledge of K
- Yet: same attack as w/ encryption is possible!
- Can be mounted even if CA requires priv-key PoP! [BM99]
 - Even if signer's id put under sig ("on-line registration attack")

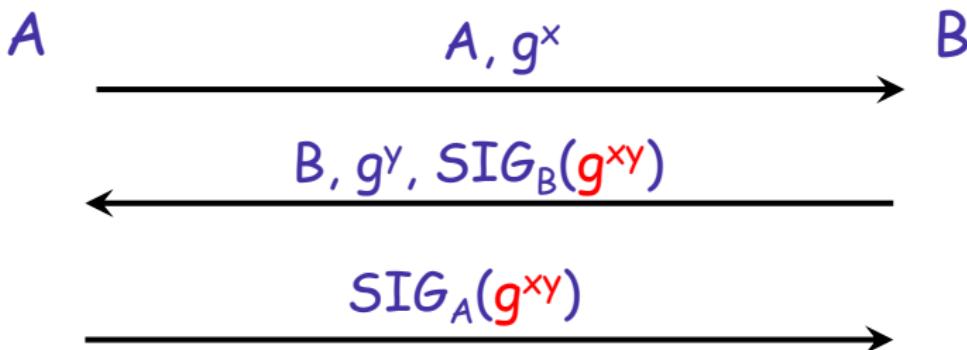
What is going on?

- The point is that “proof of knowledge” of $K=g^{xy}$ is not the issue
- What is required is:
binding the key K with the peer identities
- Which brings us to the **SIGMA** design
 - **SIGN** and **MAC**-your-own-identity!!
- And what about Photuris?
 - Yet another STS variant: Sign $K=g^{xy}$ as “proof of knowledge”; also insecure (see the SIGMA paper)
- But first another don’t-do-it lesson: Photuris

But first another don't-do-it lesson: Photuris

Photuris Protocol (basic version)

Sign g^{xy} as direct authentication of g^{xy} and proof of knowledge of g^{xy}



- **Id-Misb attack:** Eve replaces $\text{SIG}_A(g^{xy})$ with $\text{SIG}_E(g^{xy})$, possible with RSA (no need for E to register A's PK as her own)
- **SIG leaks information about g^{xy}** e.g. $H(g^{xy})$ with RSA. Breaks secure channel
 - . if keys derived as HMAC(key= g^{xy} , data) which is computable given $H(g^{xy})$
- **Small subgroup attack:** next

Photuris Protocol (basic version)

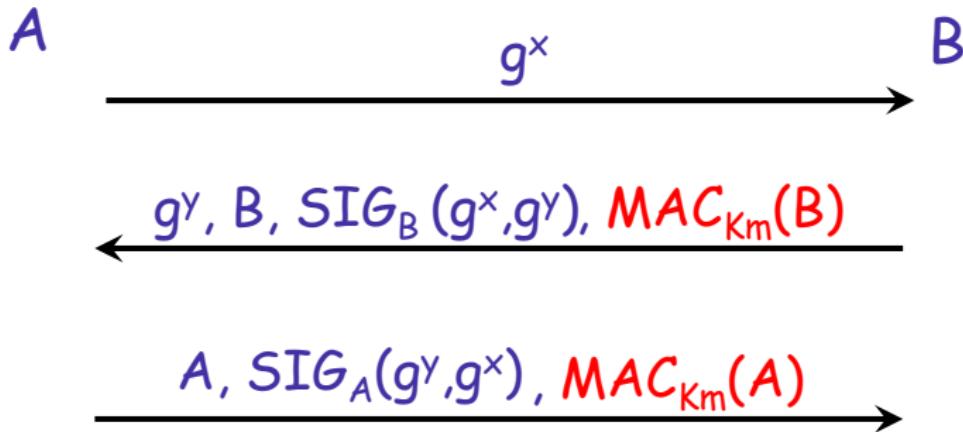


SMALL SUBGROUP ATTACK:

- Assume g in \mathbb{Z}_p and $p-1$ has a small divisor s (i.e. $p=s\cdot t+1$, e.g. $s=3$)
 - Attacker replaces g^x with $(g^x)^\dagger$ and g^y with $(g^y)^\dagger$
 - A and B compute same key $K=g^{xy\dagger}$
 - But K now has order s , hence it only has s possible values! $(g^\dagger, g^{2\dagger}, \dots, g^{s\dagger})$
- Adding g^x, g^y under sig solves the small group attack but not the other attacks*

The SIGMA Protocols

SIGMA: Basic Version

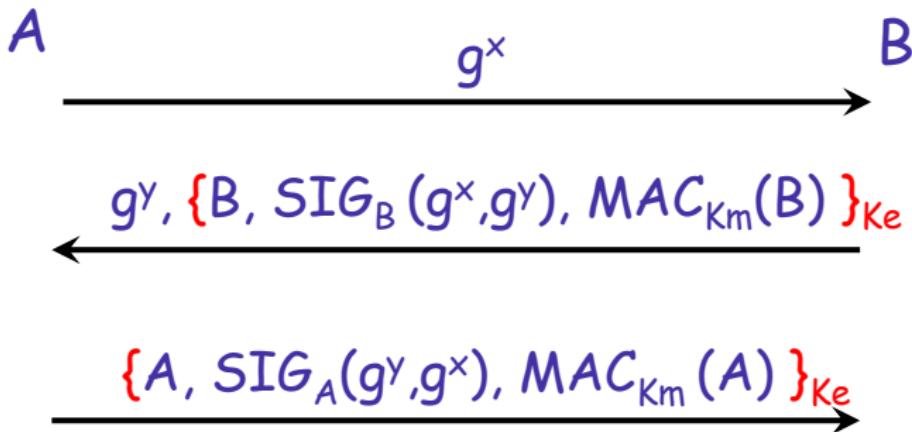


* K_m (and session key) derived from g^{xy} via a kdf

SIG and MAC: complementary roles (mitm and binding, resp)

Does not require knowing the peer's id for own authentication → Great for id protection (& deniable)

SIGMA-I: active protection of Initiator's id

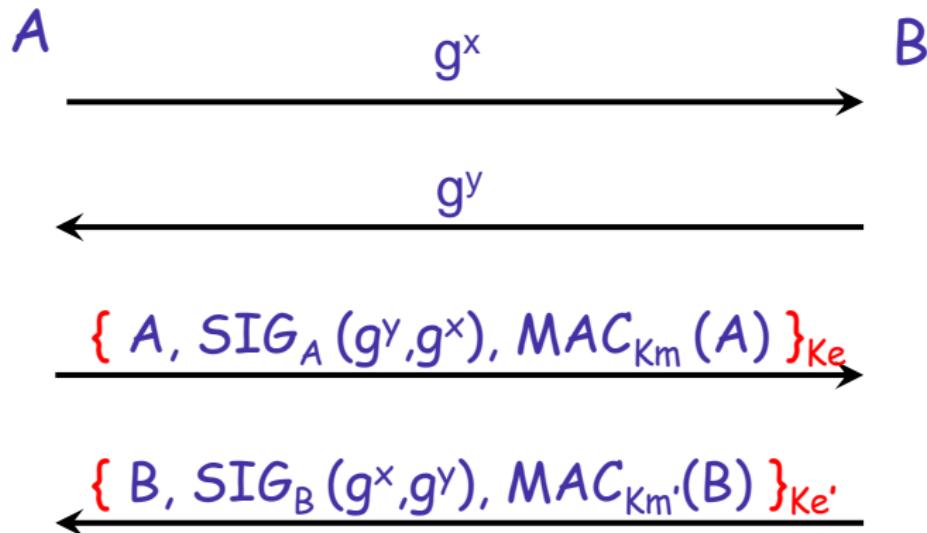


* K_e and K_m derived from g^{xy} via pseudorandom function

Responder (B) identifies first

→ Initiator's (A) id protected

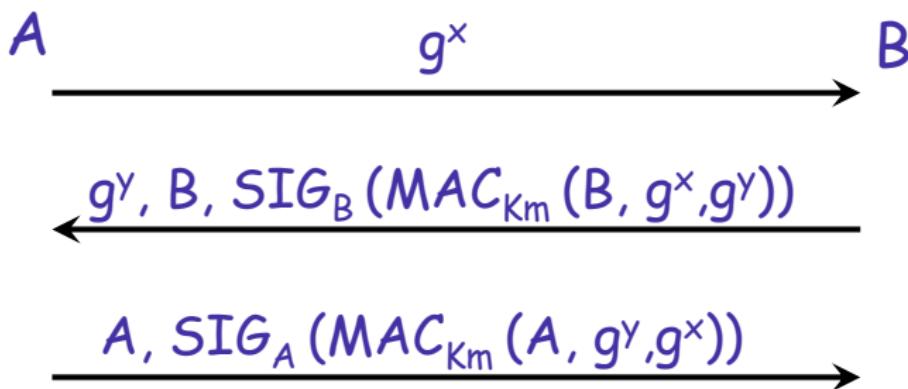
SIGMA-R: active protection of Responder's id



Note: Km , Km' and Ke , Ke' (against **reflection attack**)

IKEv1 Variant: MAC under SIG

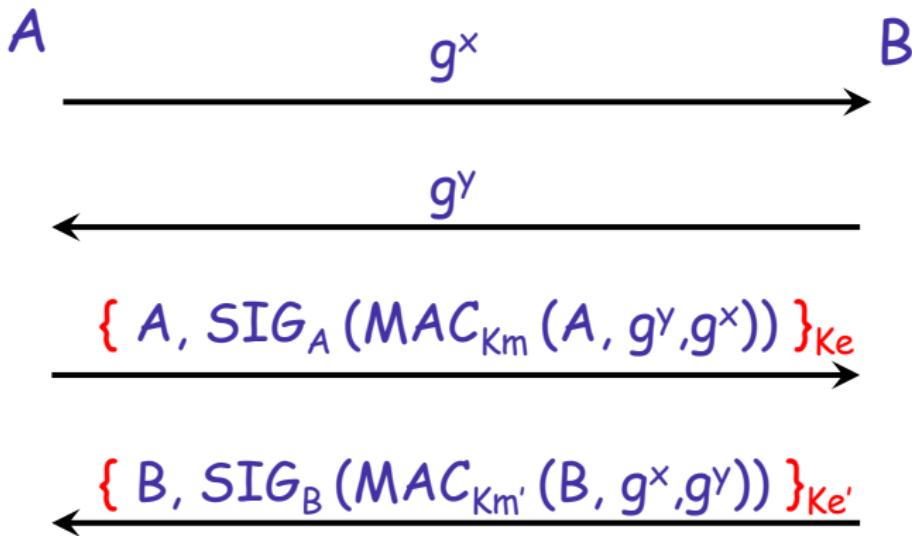
Equivalent security (just save MAC space):



→ this is IKE's "aggressive mode" (no id protect'n)

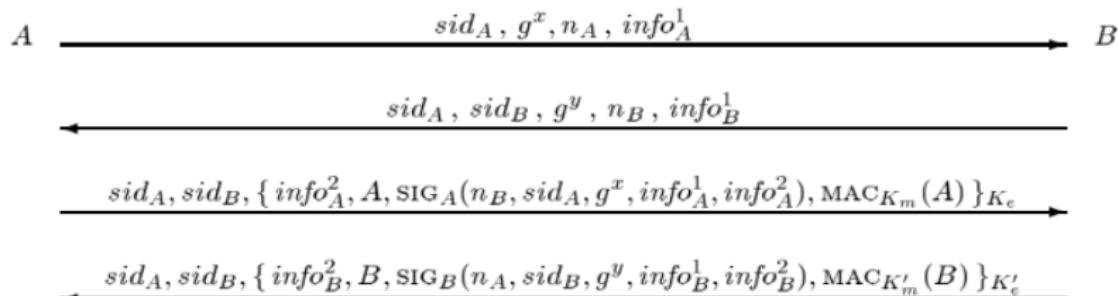
Note: $\text{MAC}(\text{SIG}(\text{id}, \dots))$ is not secure!! (STS-MAC)

IKE Main Mode



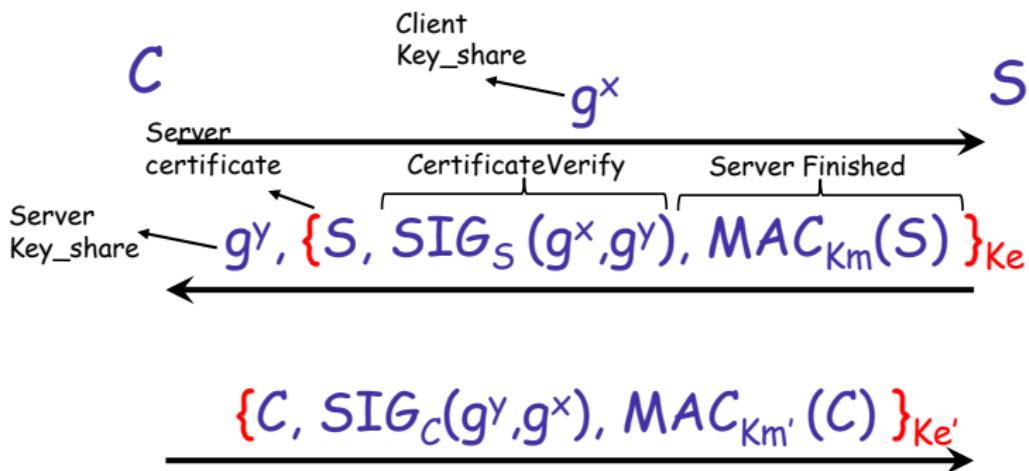
IKE v2: a slight variant - only MAC(id) under SIG

"Full pledge" SIGMA (elements missing from skeleton figures)



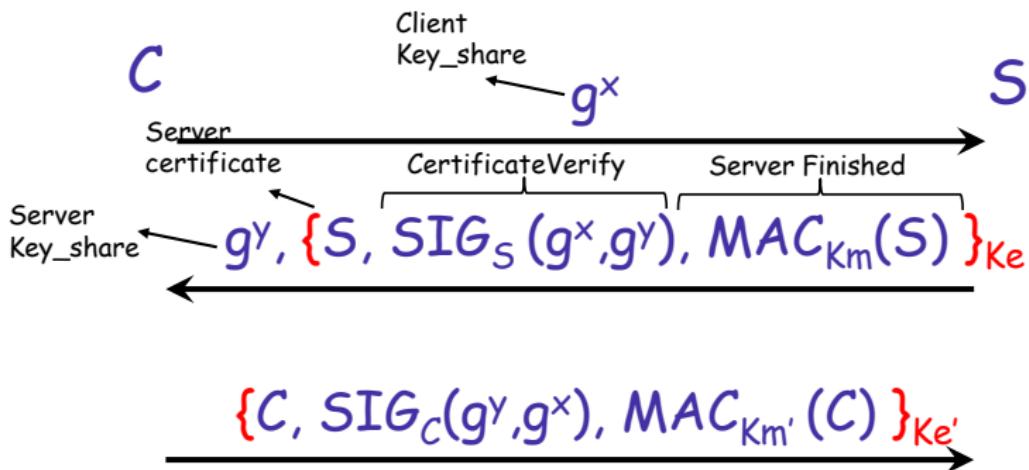
- Essential additional information (context, negotiation, etc.)
- Have separate elements for session identifiers, nonces, DH values
 - Basic principle: Don't use same element for multiple purposes
- Authenticate every element sent on the wire (identities too?)
 - Downgrade attacks

SIGMA-I → TLS 1.3



* K_e, K_e', K_m, K_m' and session key derived from g^{xy} via KDF
(handshake traffic / finished / application traffic keys)

SIGMA-I \rightarrow TLS 1.3



- PLUS: hello msg: nonce + negotiation;
transcript-hash under signature and MAC
- LESS: C, SIG_C for server-only authentication

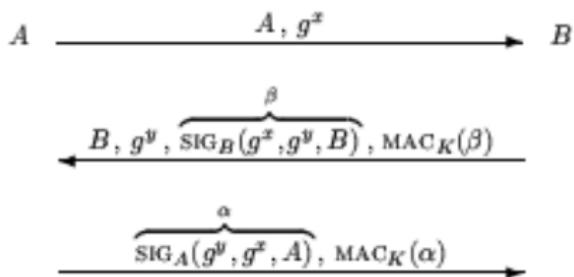
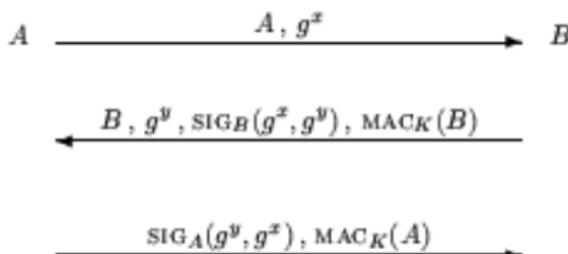
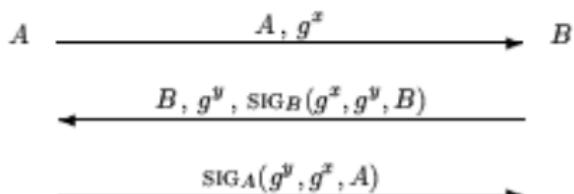
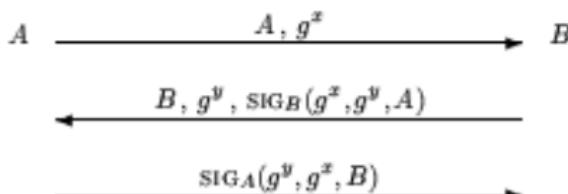
SIGMA Summary

- SIGMA suitable for most applications requiring a Diffie-Hellman key exchange:
 - Simple and efficient (minimizes msgs and comput'n)
 - No over-design (nor under-design)
 - With or without ID Protection
 - Standardized: core key-exchange protocol for both IKEv1 and IKEv2, now also TLS 1.3
 - The "off-the-record communication" [Goldberg-Borisov] (use of deniability - proven in [DGK])

But is SIGMA Secure?

- Proof in Canetti-K Crypto'02
 - Formal proof: each element is essential
 - e.g., $SIG(MAC(id, \dots))$ vs. $(SIG(id, \dots), MAC(SIG(id, \dots)))$
 - Implies secure channels
 - Secure composition (universal composability, UC)
 - From theory to practice
 - Specification, implementation, details, DoS, certificates, ...
- Care with variants!!

Care with Variants



KEM as Diffie-Hellman Generalization

- KEM: Key encapsulation mechanism ("key transport")
 - $\text{KEM}(\text{pk}) \rightarrow (\text{c}, \text{K})$ $\text{De-KEM}(\text{sk}, \text{c}) \rightarrow \text{K}$
- Generic DH:
 - Alice chooses pair (sk, pk) sends pk to Bob, keeps sk
 - Bob applies $\text{KEM}(\text{pk})$, sends c to Alice, outputs K
 - Alice computes K as $\text{De-KEM}(\text{sk}, \text{c})$, outputs K
- SKEME protocol: Long-term KEM + ephemeral KEM
 - Recent examples: OPTLS (TLS 1.3 proposal), Kyber (a lattice based "post-quantum" KE proposal), dispenses with signatures

Exercise

- Is this secure (as one-sided authentication)
- $B \rightarrow A$: Nonce r
- $A \rightarrow B$: $c = \text{KEM}(\text{PK}_B), \text{Sig}_A(c, r)$
 - Session key $K = \text{De-KEM}(\text{sk}_B, c)$

Many more considerations...

- Negotiation: algorithm independ., downgrade protection
- Denial of Service protection
- Key derivation
- Secure channels, authenticated encryption
- Deniability
- ...
- Automated analysis

PROOF-DRIVEN DESIGN®

- Proof-driven design: Formal analysis as main design tool
 - No simulation or empirical evidence; universal quantifier "for all"
- Proof guides choice of mechanisms, compose them right, discern between the essential, desirable and dispensable
- Result is efficiency, simplicity, rationale, even implementation guidance!
 - Simplicity as a security feature: Keep it simple! (but not simpler)

TLS

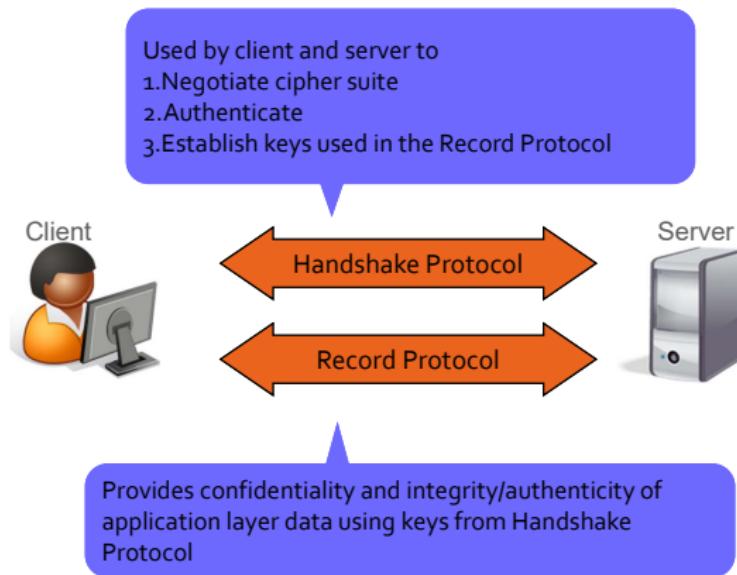
TLS Overview

- SSL = Secure Sockets Layer.
 - Developed by Netscape in mid 1990s.
 - SSLv1 broken at birth.
 - SSLv2 flawed in several ways, now IETF-deprecated (RFC 6176).
 - SSLv3 now considered broken (POODLE + RC4 attacks), but still widely supported.
- TLS = Transport Layer Security.
 - IETF-standardised version of SSL.
 - TLS 1.0 in RFC 2246 (1999).
 - TLS 1.1 in RFC 4346 (2006).
 - TLS 1.2 in RFC 5246 (2008).
 - TLS 1.3 currently in development in IETF.

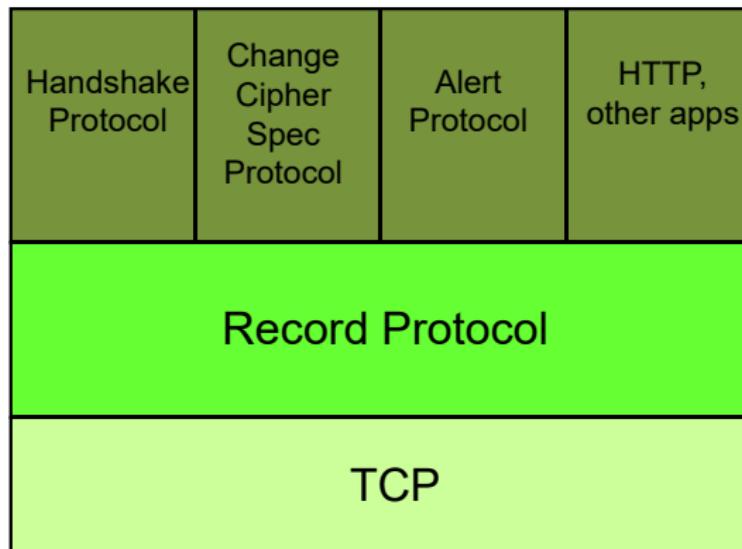
Importance of TLS

- Originally designed for secure e-commerce, now used much more widely.
 - Retail customer access to online banking facilities.
 - Access to gmail, facebook, Yahoo, etc.
 - Mobile applications, including banking apps.
 - Payment infrastructures.
- TLS has become a *de facto* secure protocol of choice.
 - Used by hundreds of millions (billions?) of people and devices every day.
 - So we should analyse it, in order to find and remove flaws.
 - Maybe we'll also find new requirements not well-reflected in the current key exchange literature?

Highly Simplified View of TLS



TLS Protocol Architecture



The TLS Ecosystem (1/3)

- Servers
 - Including managed service providers (CloudFlare, Akamai)
- Clients
 - Of all shapes and sizes
 - Web browsers to embedded devices
- Certification Authorities (CAs)
 - Of all shapes , sizes and levels of security
 - Typically 300 root CA keys in browser.
- Software vendors
 - From Google (BoringSSL) down to one-man open-source operations
 - OpenSSL somewhere in-between, once used by approx 90% of web servers.
- Hardware vendors

The TLS Ecosystem (2/3)

- TLS versions:
 - SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2, TLS 1.3 nearly complete
 - Some servers even still support SSL 2.0
- 200+ cipher suites
 - <https://www.thessprawl.org/research/tls-and-ssl-cipher-suites>
 - Some very common, e.g.
 - TLS_RSA_WITH_AES_128_CBC_SHA256
 - Some highly esoteric, e.g.
 - TLS_KRB5_WITH_3DES_EDE_CBC_MD5
 - Some offering no security:
 - TLS_NULL_WITH_NULL_NULL !
- TLS extensions
 - Too numerous to mention.
- DTLS

The TLS Ecosystem (3/3)

- IETF TLS Working Group
 - Also IETF UTA Working Group (UTA = Using TLS in Applications)
 - And CFRG (Crypto Forum Research Group)
- Growing community of researchers
 - Finding attacks and building security proofs
 - Analysis of TLS 1.3
 - TRON workshop and follow-ups, including CWTLS1.3 co-located with CRYPTO 2018.
- The TLS ecosystem has become very complex and vibrant.

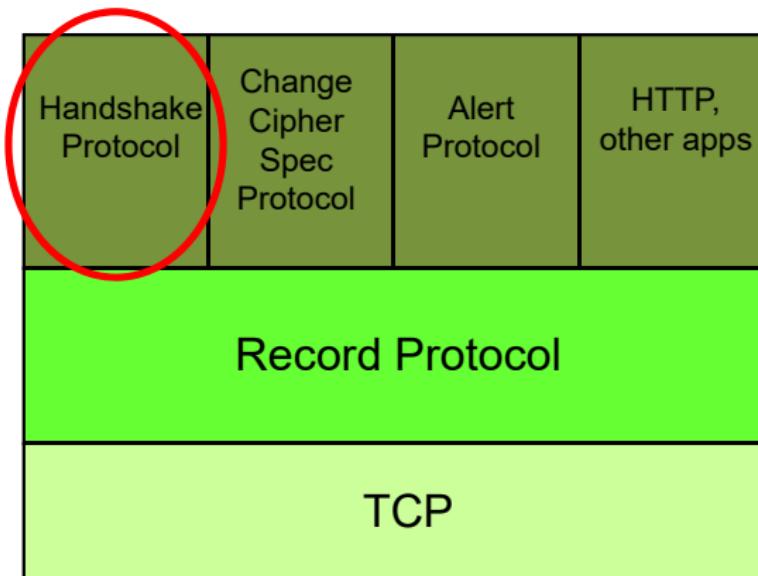
TLS Record Protocol

Redacted



TLS Handshake Protocol

TLS Protocol Architecture



TLS Handshake Protocol – Goals

Establishes keys (and IVs) needed by the Record Protocol.

Via establishment of the TLS master_secret and subsequent key derivation.

Provides authentication of server (usually) and client (rarely)

Using public key cryptography supported by digital certificates.

Or pre-shared key (less commonly).

Protects negotiation of all cryptographic parameters.

SSL/TLS version number.

Encryption and hash algorithms.

Authentication and key establishment methods.

To prevent version rollback and cipher suite downgrade attacks.

TLS Handshake Protocol – Key Establishment

TLS supports several key establishment mechanisms.

Method used is negotiated during the Handshake Protocol itself.

- Client sends list of *cipher suites* it supports in ClientHello; server selects one and tells client in ServerHello.
- e.g. TLS_RSA_WITH_AES_256_CBC_SHA256
- e.g. TLS_ECDHE_RSA_WITH_RC4_128_SHA
- cipher suites are encoded as 2-byte values.

A common choice is RSA encryption.

- Server sends RSA public key and certificate chain after ServerHello.
- Client chooses pre_master_secret, encrypts using public RSA key of server, sends to server.
- RSA encryption is based on PKCS#1 v1.5 padding method.

TLS Handshake Protocol – RSA-based Key Establishment (Simplified)

Client

Server

ClientHello (TLS_RSA_WITH_AES_256_CBC_SHA256)



ServerHello, Cert, ServerHelloDone



1. Check ServerCert
2. Extract PubK from ServerCert
3. Select random pre_master_secret
4. Compute $\text{Enc}_{\text{PubK}}(\text{pre_master_secret})$

ClientKeyExchange: $\text{Enc}_{\text{PubK}}(\text{pre_master_secret})$



Decrypt to find
pre_master_secret

TLS Handshake Protocol – Ephemeral DH-based Key Establishment (Simplified)

Client

ClientHello (TLS_DHE_RSA_WITH_RC4_128_SHA)

Server

ServerHello, Cert, ServerKeyExchange, ServerHelloDone

- 1.Check Cert
- 2.Extract PubK from ServerCert
- 3.Use PubK to check server signature
- 4.Choose y , compute g^y , $(g^x)^y$

$p, g, g^x,$
RSAsig(nonces, params)

ClientKeyExchange: g^y

pre_master_secret:
 $(g^y)^x$

TLS Handshake Protocol – Other Key Establishment Options

Static Diffie-Hellman

- Server certificate contains DH parameters (group, generator g) and static DH value g^x .
- Client chooses y , computes g^y and sends to server.

$$\text{pre_master_secret} = g^{xy}.$$

Anonymous Diffie-Hellman

- Each side sends Diffie-Hellman values in group chosen by server, but no authentication of these values.
- Vulnerable to man-in-middle attacks.

FF-DH-based Cipher Suites for TLS

- Originally, only finite-field DH was available in TLS; ECC came later.
- Recall: server chooses and sends parameters (p, g, g^x) .
- Parameters are actually under-specified: it is hard for client to verify that:
 - p is prime.
 - g has large prime order dividing $p-1$.
 - g^x is indeed a power of g , and not in some other subgroup.
- Most implementations perform only rudimentary checks.
- Issues are meliorated to some extent by use of safe-primes ($p = 2q+1$ with q prime), but also the source of some attacks, e.g. Lim-Lee small sub-group attacks.
- See Valenta *et al.* (NDSS 2017) for more details.

ECC-based Cipher Suites for TLS

- ECC-based cipher suites for TLS were first defined in RFC 4492 (Blake-Wilson *et al.*, 2006).
- Negotiated via **TLS extensions** sent in ClientHello/ServerHello messages.
- 25 different curves + 3 point formats defined in RFC 4492, along with the ability to negotiate bespoke curve.
- Many curves taken from NIST and ANSI standards, e.g. NISTp256.
- Dozens of new cipher suites (56 with “ECDH(E)”, 24 with “ECDSA”) including “SHOULD support” for:
 - TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
 - TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
 - TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
 - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

TLS Handshake Protocol – Key Establishment Notes

- Typical ClientHello offers many different cipher suites, choice of which to use is made by server.
- ClientHello and ServerHello also contain 32-byte nonces (28-byte random values + 4-byte time encoding).
- These are signed by the server in DH-based cipher suites, and involved in key derivation.
- Important for security – informally, preventing session replay attacks forcing reuse of session keys.

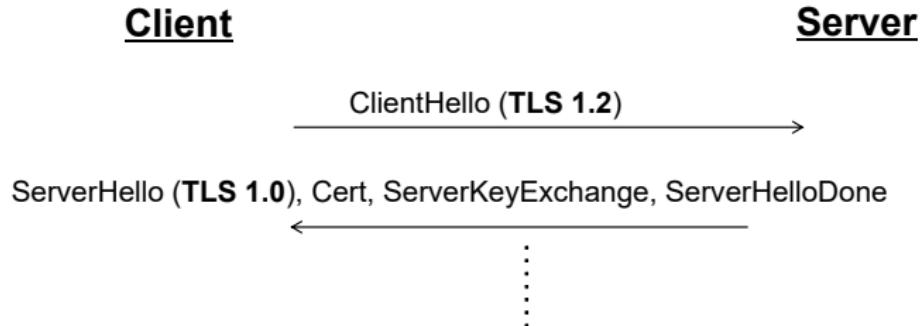
TLS Handshake Protocol – Key Establishment Notes

- ClientHello also offers SSL/TLS version number; server replies with its choice.

Semantics: client: I support up to version x ;
server: I will use version $y \leq x$.

 - Legacy servers do not implement this correctly, simply failing if they don't support version x .
 - Typical client behaviour: try again with lower version in a fresh handshake, with no memory of offers in previous handshakes carried over.
 - Security consequence: an active MITM can force client/server to roll back to **lowest** SSL/TLS version they are both willing to use!
 - POODLE attack exploits this to roll back to SSL3 and then perform Moeller attack on SSLv3 padding (see later).
 - The problem has been reanimated with the coming of TLS 1.3.

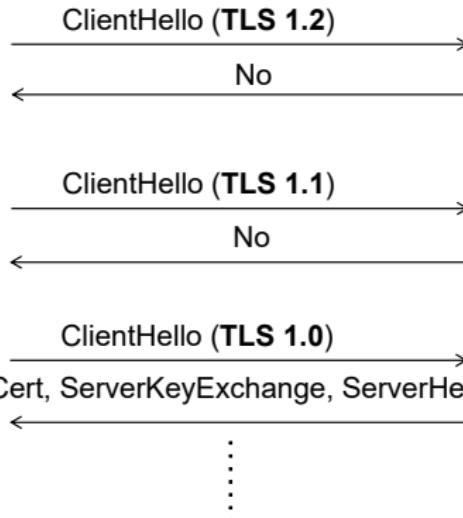
TLS Handshake Protocol Version Negotiation – Ideal World



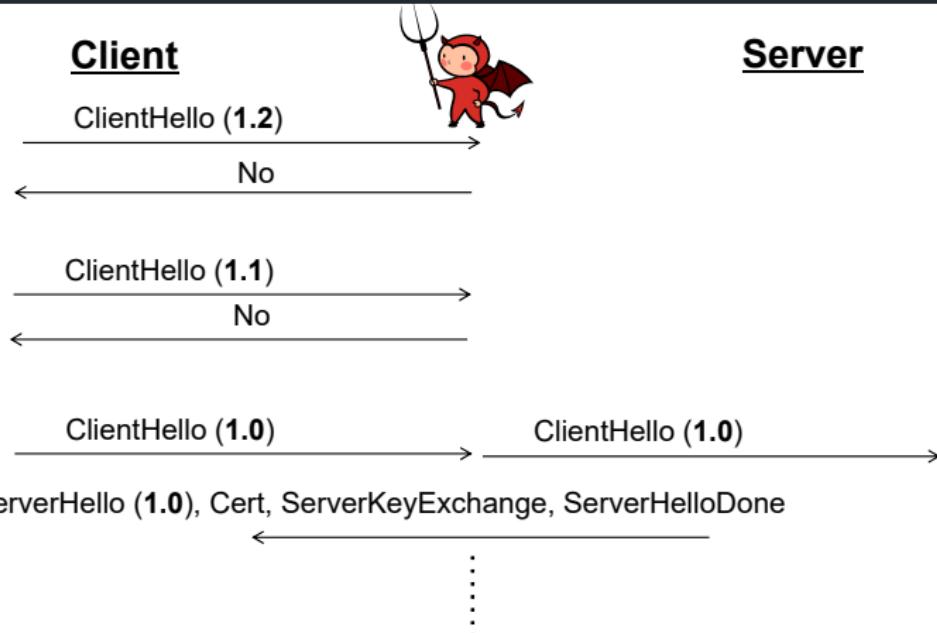
TLS Handshake Protocol Version Negotiation – Real World (Version Intolerance)

Client

Server

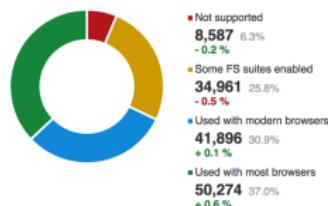
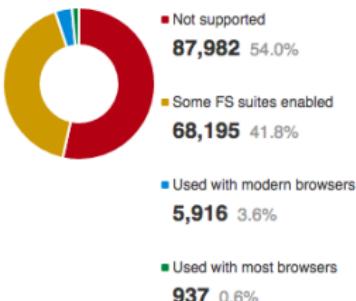


TLS Handshake Protocol Version Negotiation – Real World Attack



TLS Handshake Protocol – Forward Security?

- An attacker who learns the RSA private key can decrypt old sessions and passively eavesdrop on all future RSA-based sessions!
- A well-known issue (lack of forward security), but given prominence by the Snowden revelations.
- This and performance benefits has driven an increased usage of forward-secure, Diffie-Hellman-based cipher suites over the last few years.

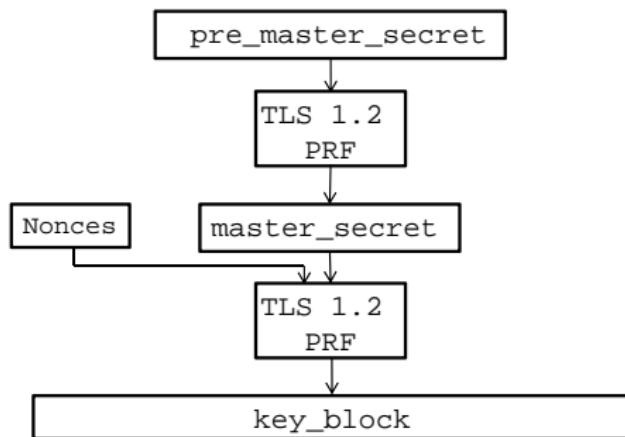


TLS Handshake Protocol – Reliance on Randomness

- An attacker who can predict a client's choice of pms or client/server DH private value can passively eavesdrop on all sessions!
 - And nonces in Hello messages may already leak information about state of client or server PRNG.
 - Hence backdoored PRNGs present a serious risk to TLS security: they may allow recovery of future PRNG output from observed output(s).
 - See Checkoway et al. (USENIX Security 2014) for extended analysis of exploitability of Dual EC PRNG in the TLS context.
- Relatedly, many server implementations default to using a "repeated ephemeral" value.
- cf. CVE-2016-0701:

OpenSSL provides the option SSL_OP_SINGLE_DH_USE for ephemeral DH (DHE) in TLS. It is not on by default.
- Hence one-time server compromise would undermine the security of many client sessions.

TLS Key Derivation



TLS Key Derivation

Keys used by MAC and encryption algorithms in the Record Protocol are derived from `pre_master_secret` (pms):

- Derive `ms` from pms using TLS Pseudo-Random Function (PRF).
- Default PRF for TLS1.2 is built by iterating HMAC-SHA256 in a specified way; earlier versions use *ad hoc* MD5/SHA-1 combination.
- Derive `key_block` from `ms` and client/server nonces exchanged during Handshake Protocol.
- Again using the TLS PRF in TLS1.2.
- Split up `key_block` into MAC keys, encryption keys and IVs for use in Record Protocol as needed.
- NB1: neither client nor server identity is involved in key derivation, nor any cipher suite context.
- NB2: splitting up of `key_block` into components depends on cipher suite.

TLS Handshake Protocol – RSA-based Authentication?

Client

Server

ClientHello (TLS_RSA_WITH_AES_256_CBC_SHA256)



ServerHello, Cert, ServerHelloDone



1. Check ServerCert
2. Extract PubK from ServerCert
3. Select random pms
4. Compute $\text{Enc}_{\text{PubK}}(\text{pms})$

ClientKeyExchange: $\text{Enc}_{\text{PubK}}(\text{pms})$



Decrypt to find pms

TLS Handshake Protocol – RSA-based Authentication

Client

Server

ClientHello (TLS_RSA_WITH_AES_256_CBC_SHA256)



ServerHello, Cert, ServerHelloDone



ClientKeyExchange: $\text{Enc}_{\text{PubK}}(\text{pms})$



1. Decrypt to find pms
2. Derive ms
3. Compute ServerFinished = PRF(ms,transcript)

ServerFinished



1. Derive ms
2. Compute ServerFinished' = PRF(ms,transcript)
3. Compare to received version

Server authenticated to Client by proving its ability to decrypt using Server's private key

TLS Handshake Protocol – Authentication for Ephemeral DH-based Key Establishment

Client

Server

ClientHello (TLS_DHE_RSA_WITH_RC4_128_SHA)

ServerHello, Cert, ServerKeyExchange, ServerHelloDone

- 1.Check Cert
- 2.Extract PubK from ServerCert
- 3.Use PubK to check server signature
- 4.Choose y, compute g^y , $(g^x)^y$

$p, g, g^x,$
RSAsig(nonces, params)

ClientKeyEx

Server authenticated to client
by showing its ability to sign
client nonce using the Server's
private key

ServerHelloDone

master_secret:
 $(g^y)^x$

TLS Handshake Protocol – Authentication

TLS supports several different entity authentication mechanisms for clients and servers.

Method used is negotiated along with key exchange method during the Handshake Protocol itself.

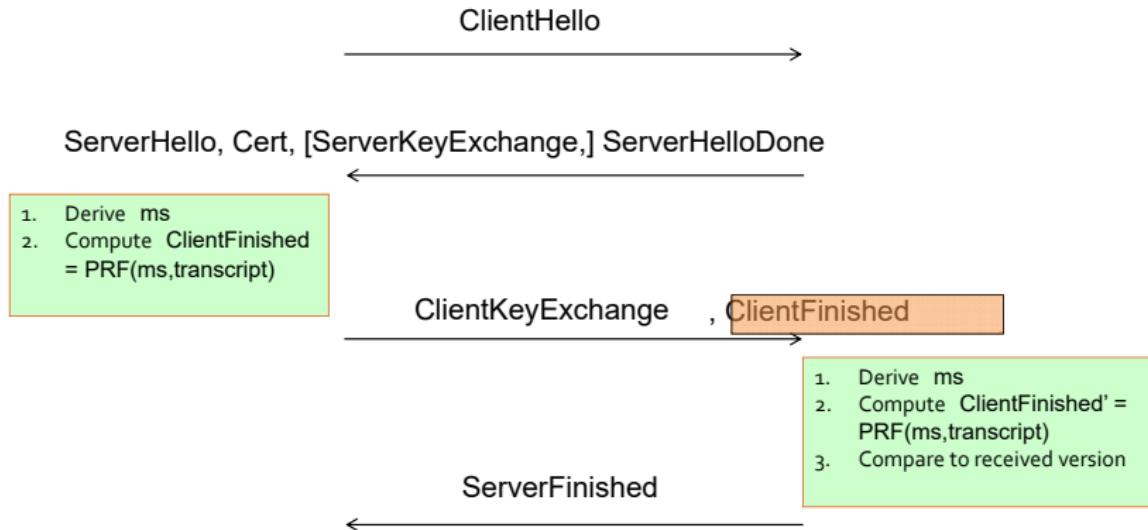
RSA: Ability of server to decrypt pms using its private key, derive ms from pms and then generate correct PRF value in ServerFinished message.

DHE/ECDHE: Ability of server to sign ClientNonce using its private key.

TLS Handshake Protocol – ClientFinished

Client

Server



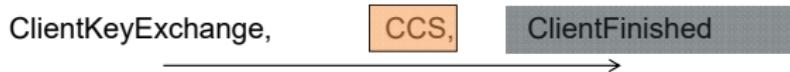
TLS Handshake Protocol – Finished Messages

- TLS Finished messages enable each side to check that both views of the Handshake Protocol are the same.
- Computed as $\text{PRF}(\text{ms}, \text{transcript})$ where transcript = sender's view of all protocol messages sent and received up to *this* point.
- Compared by recipient to expected value; protocol aborts if mismatch is observed.
- Designed to prevent version rollback and cipher suite downgrade attacks.
 - Attacker attempts to manipulate client/server view of cipher suite(s) accepted/offered, or of version offered/accepted.
 - Ineffective if attacker can compute ms during protocol run.

TLS Handshake Protocol – ChangeCipherSpec

Client

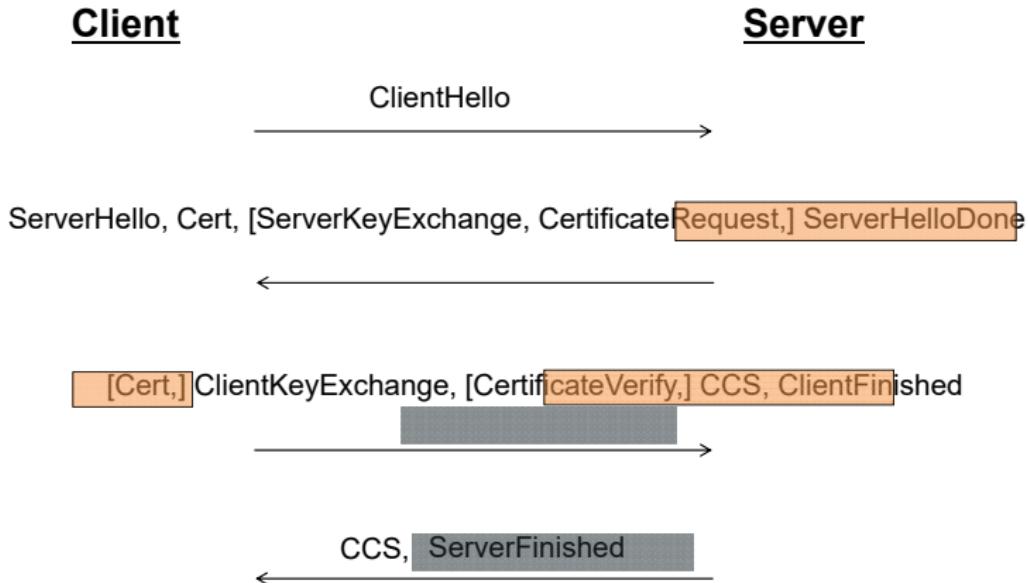
Server



TLS Handshake Protocol – CCS Messages

- ChangeCipherSpec messages enable parties to inform each other that they are switching to the recently agreed keys in the Record Protocol.
- Here, this means that all subsequent messages are protected using the agreed cipher suite (e.g. AES_256_CBC_SHA256).
- Not part of the Handshake Protocol, so not included in transcripts when computing Finished messages.

TLS Handshake Protocol – Client Authentication



TLS Handshake Protocol – Client Authentication

- Client authentication is optional and rarely used in the web setting.
- Server requests client's certificate in its **Hello** message.
- Client responds with:
 - Cert: client's certificate (chain).
 - CertificateVerify: signature on protocol transcript up to this point.
 - Notice the misnomers!

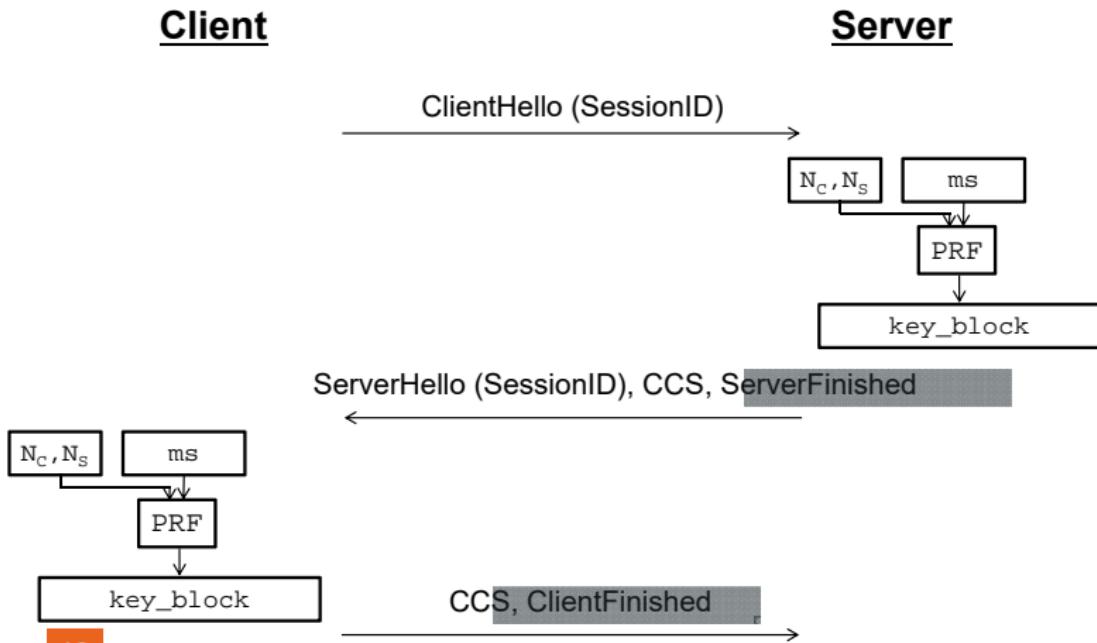
TLS Handshake Protocol – Renegotiation

- **Renegotiation** allows re-keying and change of cipher suite during a session.
 - For example, to force strong client-side authentication before access to a particular resource on the server is allowed.
 - Or to publicly negotiate a weak cipher suite and then upgrade to a stronger one over an encrypted channel.
- Initiated by client sending **ClientHello** or server sending **ServerHelloRequest**.
 - Followed by full run of Handshake Protocol.
 - Protocol is run over the existing Record Protocol, so receives its protection.

TLS Handshake Protocol – Session Resumption

- **Session resumption** allows authentication and shared secrets to be reused across multiple, parallel *connections* in a single session.
- E.g., allows fetching multiple resources from same website without re-doing full, expensive Handshake Protocol.
- Client and Server quote existing SessionID and exchange fresh nonces.
- Also enabled by use of session ticket mechanism, RFC 5077.
 - Uses a TLS extension to signal/transmit a cryptographic “blob” from server to client, carrying session state.

TLS Handshake Protocol – Session Resumption



TLS Sessions and Connections

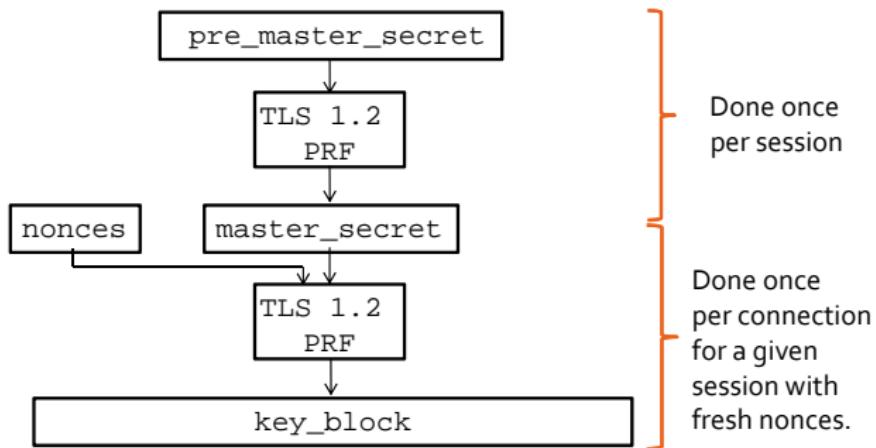
Session concept:

- Sessions are created by the Handshake Protocol.
- Session state defined by session ID and set of cryptographic parameters (encryption and hash algorithm, master secret, certificates) negotiated in Handshake Protocol.
- Each session can carry multiple **parallel connections**.

Connection concept:

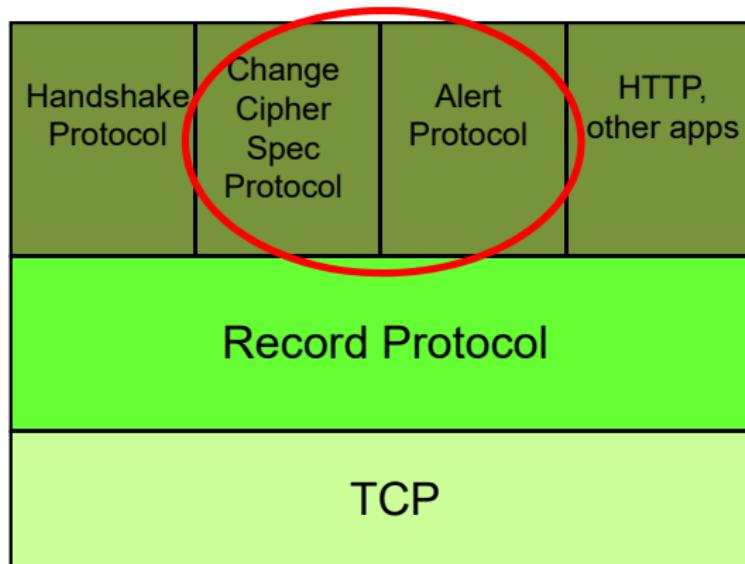
- Keys for multiple connections are derived from a single **ms** created during one run of the full Handshake Protocol.
- Session resumption Handshake Protocol runs exchange new nonces.
- These nonces are combined with existing **ms** to derive keys for each new connection.
- Avoids repeated use of expensive Handshake Protocol.
- Each TLS connection corresponds to a different TCP connection.

TLS Key Derivation and Sessions/Connections



Other TLS Protocols

TLS Protocol Architecture



Other TLS Protocols

Alert protocol.

- Management of SSL/TLS connections and sessions, error messages.
- Fatal errors and warnings.
- Defined actions to ensure clean session termination by both client and server.

Change cipher spec protocol.

- Technically not part of Handshake Protocol.
- Used to indicate that entity is changing to recently agreed cipher suite.

Both protocols run over Record Protocol (so are peers of Handshake Protocol).

TLS Extensions

Many *extensions* to TLS exist.

Allows extended capabilities and security features.

Examples:

- Renegotiation Indicator Extension (RIE), RFC 5746.
- Application layer protocol negotiation (ALPN), RFC 7301.
- Authorization Extension, RFC 5878.
- Server Name Indication, Maximum Fragment Length Negotiation, Truncated HMAC, etc, RFC 6066.

TLS Complexity

- Recall simplistic view of TLS:
 - Handshake Protocol followed by Record Protocol.
- Reality is much more complex:
 - Initial Handshake Protocol over Record Protocol with no keys.
 - Change Cipher Spec. Protocol message, switch on new keys.
 - Completion of Handshake via exchange of Finished messages, now running over keyed Record Protocol.
 - Followed by arbitrary sequences of Session Resumption and Renegotiation runs.
 - Most of this activity is hidden from applications.
- This complexity has turned out to have negative security consequences.

TLS Handshake Protocol Security Issues

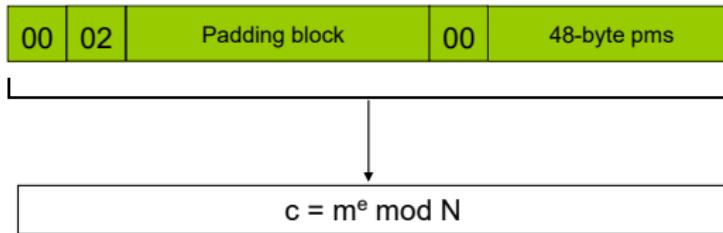
Some TLS Handshake Protocol Security Issues

- Bleichenbacher attack (1998) on PKCS#1 v1.5 padding used for RSA encryption in Handshake protocol.
 - Patched by making it hard to distinguish error messages, but attack rebooted in various ways over the years.
 - Including DROWN attack in 2016, exploiting public key reuse between SSLv2 and other versions of SSL/TLS, and extensive legacy support for SSLv2 in servers.
- Attacks exploiting continued support for weak “export-grade” cipher suites: FREAK and LOGJAM (2015).
- Attacks exploiting renegotiation and resumption: renegotiation attack (2009), triple handshake attack (2014).
- Implementation flaws of various kinds.

Bleichenbacher's Attack

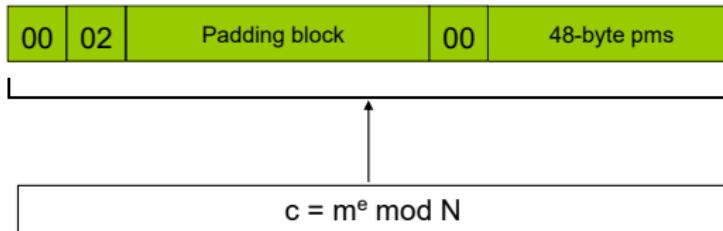
- We begin with Bleichenbacher's attack on RSA encryption used in TLS (C'98).
- This attack exploits the fact that RSA encryption scheme used in TLS (PKCS#1 v1.5) is **not** CCA secure.
- It recovers the TLS `pre_master_secret` (pms) for a target session using roughly 2^{20} interactions with server.

PKCS#1 v1.5, block type 2



- Plaintext must begin with “oo 02” bytes.
- Padding block consists of *at least* 8 non-zero bytes.
- Should be terminated by “oo” byte.
- Last 48 bytes are used as pms.
 - Additional complication: most significant two bytes are set to client TLS version.

PKCS#1 v1.5, block type 2



Think about sanity checking m after applying RSA decryption operation:

- Check for "00 02"?
- Check for at least 8 non-zero padding bytes or just *some* non-zero bytes?
- Check for a 00-byte? Or just extract last 48 bytes?
- Demand 00-byte to be in exactly the right position?
- Check for TLS version number?

Bleichenbacher's Attack

- Exact decryption processing for RSA is not specified in the RFCs.
- Different implementations exhibit different behaviours.
- To simplify matters: suppose that we have an oracle that on input c outputs whether $x := c^d \bmod N$ begins with byte pattern “00 02”.
- If oracle output is “yes”, then we have an inequality:

$$2B \leq x \bmod N < 3B$$

where $B = 2^{8(k-2)}$ and k is the number of bytes in modulus N .

Bleichenbacher's Attack

- Suppose attacker records c^* , the RSA ciphertext encrypting the unknown pms for a target session.
- Attacker calls the “oo o2” oracle on many, carefully selected inputs of the form $s^e c^* \bmod N$.
- Each “yes” output gives an inequality of the form:

$$2B \leq s^e x \bmod N < 3B$$

where s is known and x encodes pms .

- By analysing many responses from the oracle, the attacker can eventually reconstruct x and thence pms .
- Roughly 2^{20} oracle queries are needed.

Bleichenbacher's Attack

In the TLS context:

The required “oo o2” oracle was obtained using error messages arising from server processing of attacker-generated ClientKeyExchange messages.

Countermeasures?

- Switch to using CCA-secure variant of RSA encryption, e.g. RSA-OAEP (cannot create “related” ciphertexts that are valid).
- Add protocol-specific countermeasures.

Bleichenbacher and TLS1.0 (1999)

TLS 1.0 was published in RFC 2246, Jan 1999, shortly after adoption of RSA-OAEP into PKCS#1v2.0.

TLS 1.0 still uses PKCS#1v1.5, despite Bleichenbacher's attack:

The best way to avoid vulnerability to this attack is to treat incorrectly formatted messages in a manner indistinguishable from correctly formatted RSA blocks. Thus, when it receives an incorrectly formatted RSA block, a server should generate a random 48-byte value and proceed using it as the premaster secret. Thus, the server will act identically whether the received RSA block is correctly encoded or not.

TLS 1.1, RFC 4346 (2006)

[PKCS1B] defines a newer version of PKCS#1 encoding that is more secure against the Bleichenbacher attack. However, for maximal compatibility with TLS 1.0, TLS 1.1 retains the original encoding. No variants of the Bleichenbacher attack are known to exist provided that the above recommendations are followed.

Over-optimistic: several implementations still get it wrong, and there's now a long literature of Bleichenbacher-style attacks against RSA implementations (not just in TLS):

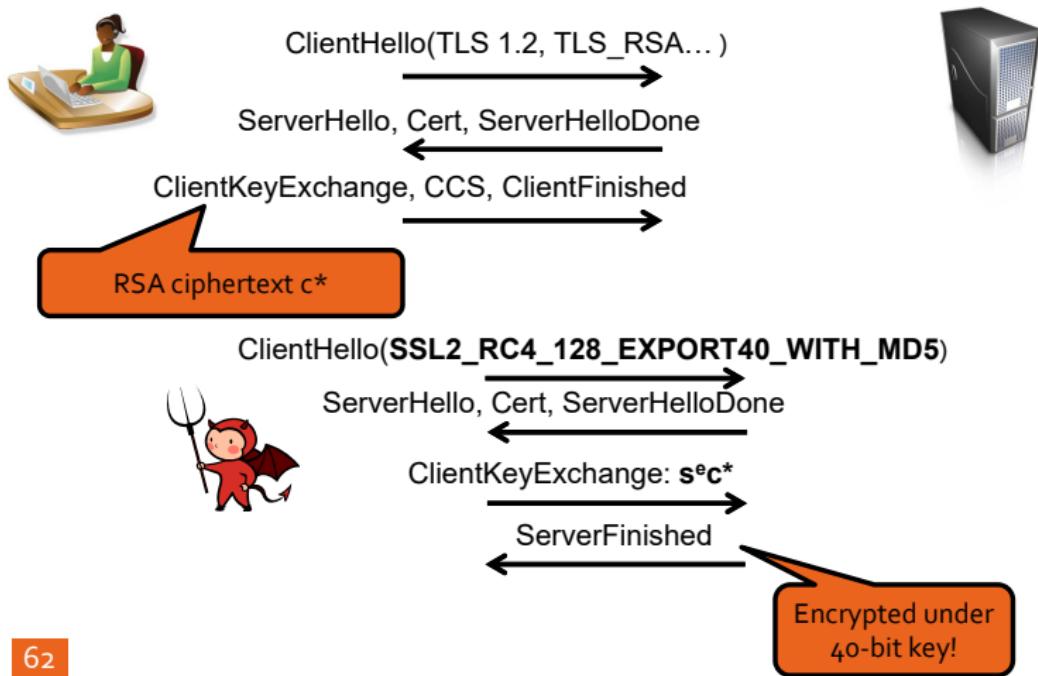
- Bardou et al. (Crypto 2012), Jager et al. (Esorics 2012), DROWN (Aviram et al., USENIX 2016), ROBOT (Böck et al., 2017).

DROWN Attack (Aviram et al., USENIX'16)

Attack scenario:

- Server supports SSLv2 and uses the same RSA key for SSLv2 and later versions of SSL/TLS
 - Surprisingly large number of servers: circa 8% of Alexa top 150k servers in July 2016 (SSL pulse)
 - Most servers don't provide a facility to provide different key for different SSL/TLS versions anyway.
- Client has absolutely no intention to use SSLv2.

DROWN Attack (Aviram et al., USENIX'16)



DROWN Attack (Aviram et al., USENIX'16)

- Standard Bleichenbacher countermeasure: if RSA decryption of c^* fails, then choose a random master secret K and carry on with the protocol.
- Send s^{ec^*} twice in two consecutive SSLv2 handshakes:
 - If s^{ec^*} is invalid, we get two ServerVerify messages encrypted under two different keys.
 - If s^{ec^*} is valid, then we get two ServerVerify messages encrypted under the same key.
- But the encryption key is “only” 40-bits in size, and the plaintext is partly known.
- Perform two 40-bit key searches and compare keys to find out if s^{ec^*} was valid or invalid.
- This provides an expensive oracle for carrying out Bleichenbacher’s attack.

DROWN Attack (Aviram et al., USENIX'16)

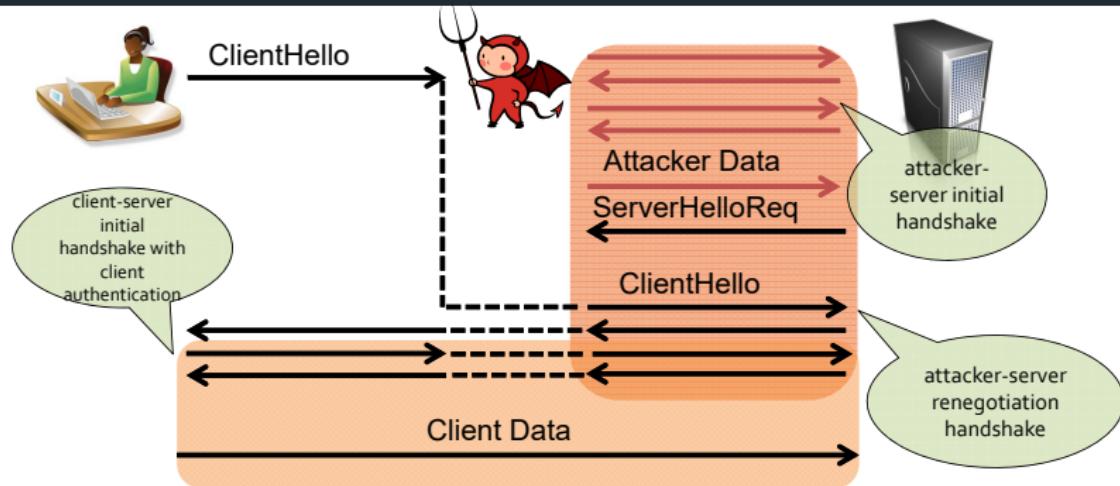
- Roughly 10,000 SSLv2 handshakes are needed for the attack, and the attack works for (roughly) 1 in every 1000 TLS handshakes.
- Support for legacy 40-bit algorithms in SSLv2 + bugs in OpenSSL implementation make it feasible to extract plaintext underlying c*.
- Cost is 2^{50} trial decryptions (without OpenSSL bugs), but under a minute for “special DROWN” (with OpenSSL bugs).
- **This is a cross-version (or cross-cipher suite) attack, made possible because of support for old versions/algorithms and key re-use across versions.**

More Recent TLS Handshake Protocol Attacks

Up until 2009, the TLS Handshake Protocol survived relatively unscathed.

Notable exception: Bleichenbacher's attack on RSA encryption used in TLS as discussed above.

Renegotiation Attack (Ray and Dispensa, Rex, 2009)



Client view: single handshake, sends ClientData.

Server view: two handshakes, receives AttackerData||ClientData from authenticated client.

Overall effect: attacker injects AttackerData as if from trusted source.

Renegotiation

- Renegotiation attack due to Ray and Dispensa, also Rex (2009).
- Server treats data as coming from either side of client authentication as being a single unit from an authenticated source.
- TLS specification does not really say how to handle this situation.
 - Flush buffer of received fragments upon renegotiation?
 - Signal to application that authentication status has changed?
 - Highlights lack of API specification for TLS.
- Attack addressed via Renegotiation Indication Extension (RIE), RFC 5746.
 - Include and verify information about previous handshakes in any renegotiation.
 - Could also disable renegotiation on server.

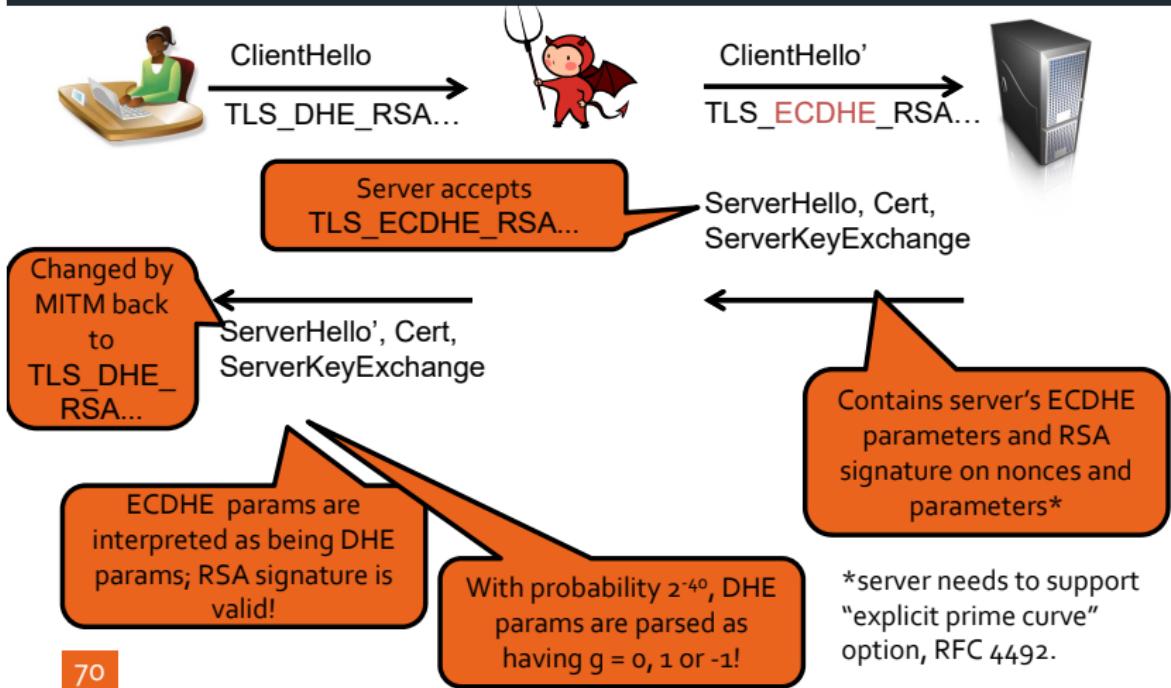
Triple Handshake Attack (Bhargavan et al, IEEE S&P'14)

- Triple Handshake attack: renegotiation attack rebooted.
- Complex attack leveraging lack of identities in key derivation + resumption + renegotiation.
 - Even first step in the attack (UKS attack) breaks certain authentication protocols relying on TLS.
- Attack highlights that RIE fix for renegotiation attack is not robust in the context of the full TLS Handshake Protocol.
 - Renegotiation status gets lost across resumptions.

Cross-cipher Suite Attacks

- Recall server signature format in `ServerKeyExchange`:
 $\text{sig}(\text{nonces}, \text{params})$
- Format of params depends on type of key exchange: mod p DH parameters or ECDH parameters.
- But *type* of parameters is not itself signed.
- Instead, it's inferred by client from the cipher suite, for which agreement is only verified later, via `Finished` messages.
- Leads to a theoretical attack due to Mavrogiannopoulos *et al.* (CCS'12).
 - Attacker switches cipher suite – ECDH for FFDH, or vice-versa.

Cross Cipher Suite Attack (Mavrogiannopoulos et al., CCS'12)



Cross Cipher Suite Attack (Mavrogiannopoulos et al, CCS 2012)

- Attack requires server to support “explicit prime curve” option (RFC 4492).
- Attack requires client to accept weak DH parameters ($g = 0, 1$ or -1).
 - Enabling MITM to compute pms and correct ServerFinished message to complete the handshake.
- Success rate can be boosted by repeatedly sending ClientHello message within TLS timeout on client (tens of seconds).
- Attack possible because server signature does not cover type of cipher suite, nor TLS extensions specifying use of ECC.

FREAK and LOGJAM Attacks

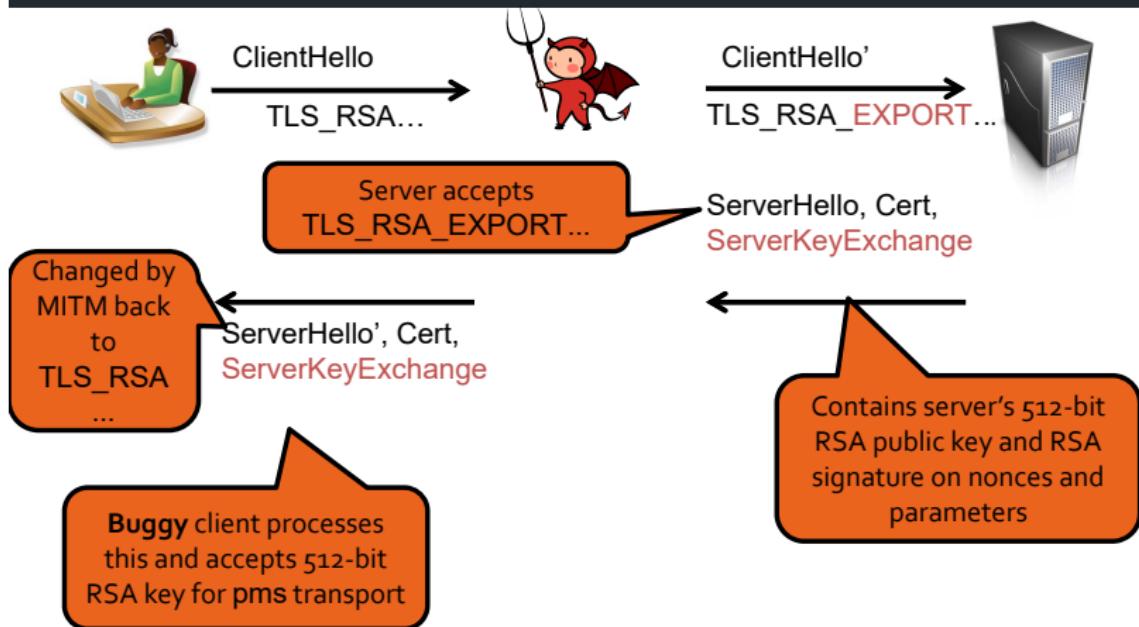
EXPORT cipher suites:

0x000003	TLS_RSA_EXPORT_WITH_RC4_40_MD5
0x000006	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
0x000008	TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
0x00000B	TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
0x00000E	TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
0x000011	TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
0x000014	TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

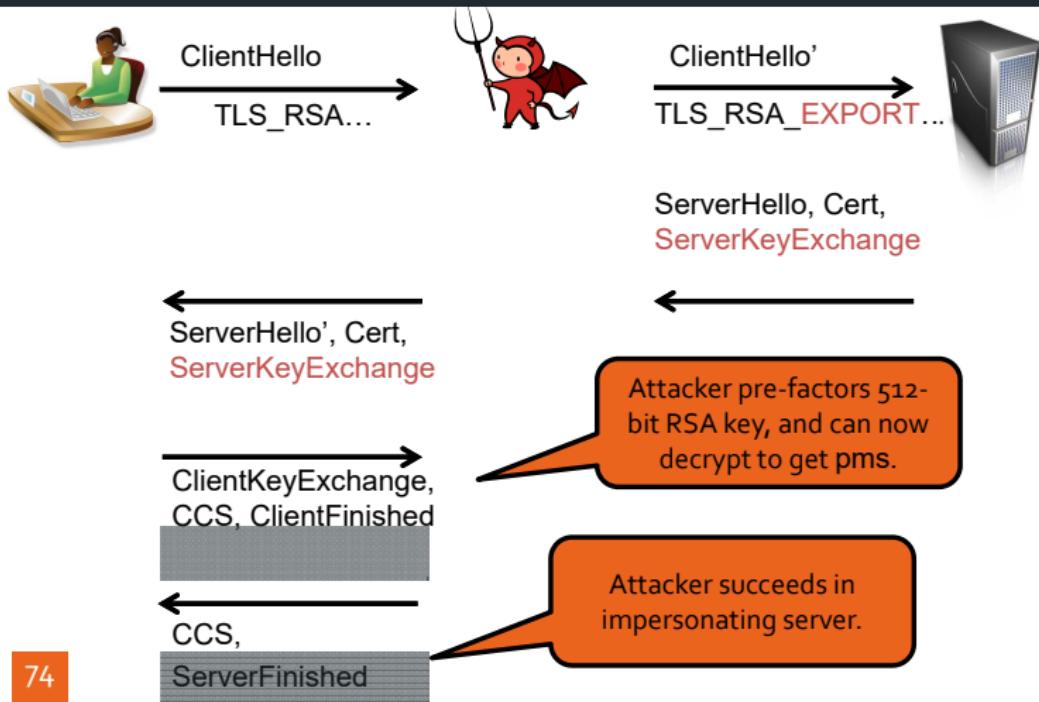
(and more)

- Introduced in the 90s in the era of export control.
- Maximum 512-bit RSA keys and 512-bit primes for DH/DHE.
- Repurpose ServerKeyExchange message to transport “ephemeral” RSA/DH/DHE keys.
- Until recently, still supported by around 25% of servers...

FREAK Attack (Beurdouche *et al.*, IEEE S&P'15)



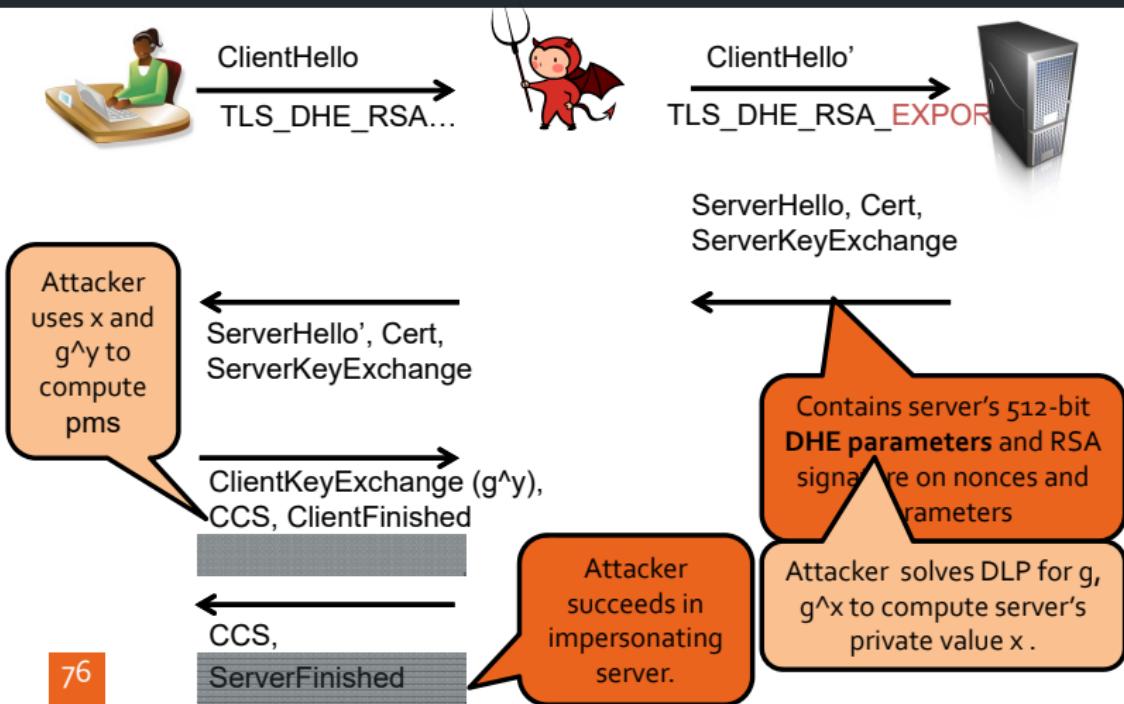
FREAK Attack (Beurdouche *et al.*, IEEE S&P'15)



FREAK Attack (Beurdouche *et al.*, IEEE S&P'15)

- Attack relies on buggy clients accepting ServerKeyExchange containing 512-bit RSA key when no such message was expected.
 - Many clients were vulnerable (<https://www.smacktls.com/>).
- Export RSA keys are meant to be ephemeral, but hard to generate RSA moduli in practice, so they were made long-lived.
- Cost of factoring 512-bit modulus: \$50 on Amazon EC2.
- Attack arises because of common code paths in implementations, coupled with state machine failures.
 - Explored in-depth in Berdouche *et al.* paper.

LOGJAM Attack (Adrian *et al.*, CCS'15)



LOGJAM Attack (Adrian *et al.*, CCS'15)

- LOGJAM = Cross-cipher suite + FREAK.
 - Active attacker changes `TLS_DHE_RSA...` to `TLS_DHE_RSA_EXPORT...`
 - Server responds with weak DH parameters signed under its RSA key.
 - Client accepts these (signature does not include cipher suite details).
 - Attacker solves 512-bit DLP before client times out.
 - Attacker can then create correct `ServerFinished` message to impersonate server.
- Difficult to perform in practice, but not impossible for three-letter agency.
 - Servers use small number of common primes p .
 - Precomputation allows each 512-bit DLP to be solved in around 90s.



Implementation Vulnerabilities



Heartbleed

- Buffer over-read vulnerability in OpenSSL implementation of DTLS Heartbeat protocol.
- High severity: remote recovery of chunks of server memory, including server private keys, private user data, etc.
- 85%+ of SSL/TLS servers rely on OpenSSL.
- Practical demonstrations of threat (e.g. Mumsnet).
- Messy disclosure in early April 2014.
- A good logo!



Certificate Processing Bugs

Many problems have been discovered in code for certificate processing.

- Fahl et al. (CCS 2012)
- Georgiev et al. (CCS 2012)
- GnuTLS bug (CVE-2014-0092)
- Apple goto fail (CVE-2014-1266)
 - Affecting Apple iOS 6.x before 6.1.6 and 7.x before 7.0.6, Apple TV 6.x before 6.0.2, and Apple OS X 10.9.x before 10.9.2.

Apple goto fail

```
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,  
                                uint8_t *signature, UInt16 signatureLen)  
{  
    OSStatus     err;  
    ...  
  
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)  
        goto fail;  
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)  
        goto fail;  
        goto fail;  
    if ((err = SSLHashSHA1.final(&hashOut, &hashOut)) != 0)  
        goto fail;  
    ...  
  
fail:  
    SSLFreeBuffer(&signedHashes);  
    SSLFreeBuffer(&hashCtx);  
    return err;  
}
```

Causes all server signature processing on client to be bypassed!

Meaning that MITM attacker can trivially impersonate **any** TLS server!

CCS Mishandling Bug (CVE 2014-0224)

- OpenSSL implementation of TLS will accept ChangeCipherSpec message at any point in the TLS Handshake.
- So MITM attacker can inject it at point of his choosing.
- Result is that TLS key derivation is carried out with a zero-length master secret.
- Leading to predictable session keys.



Invalid Curve Attacks

- Implementations fail to check that received EC point is actually on specified curve.
- Leads to **invalid curve attack** on implementations reusing ephemeral values and/or ECDH cipher suites.
- Attacker (client) sends as its DH values points $P_i = (x_i, y_i)$ in EC groups of small, co-prime orders q_i .
- Server responds by computing sP , where s is long-term secret: relies on x -coordinate-only computation depending only on b in Weierstrass form.
- `premastersecret` is then just sP , one of q_i possible values.
- Attacker can learn $s \bmod q_i$ by guessing value for `ClientFinished` and testing if server accepts.
- Attacker can reconstruct s using CRT.
- Original ideas in Biehl *et al.* (CRYPTO 2000) and Antipa *et al.* (PKC 2003).
- Shown to work in practice for TLS implementations by Jager *et al.* (ESORICS 2015).

Side Channel Attacks

- Use of public key primitives opens up many opportunities for side-channel attacks on implementations.
- Timing attacks on naïve (and not so naïve!) implementations of RSA, EC-DSA, DH, ECDH.
- A current focus is on “**Flush+Reload**” Lowest Level Cache (LLC) timing attacks.
- OpenSSL is by now quite well protected, but new attacks are still being discovered.
- Recent example: Pereida García and Brumley, USENIX 2017:
 - LLC attack on modular inversion code used in OpenSSL ECDSA, finding MSBs of ECDSA nonces, and thence ECDSA signing key via lattice attack.
- Other libraries are (probably) less-well protected.



TLS 1.3



TLS 1.3

- The TLS 1.3 specification is being developed in TLS Working Group of IETF.
- Major redesign compared to previous versions.
- Specification is now nearing completion, currently at draft 23.
- Several implementations underway, tracking changes to specification, working on inter-op.

TLS 1.3

- Main objectives for TLS 1.3:
 - Reduce latency of initial secure data communication (1-RTT and 0-RTT for resumed sessions).
 - Improve security and privacy.
 - Protocol simplification (reducing options and removing broken cipher suites).
 - **No compression, RC4, MAC-then-Encrypt, RSA key transport, custom DH and ECDH groups, renegotiation.**
 - Unifying session resumption and PSK mechanisms.
 - But continuity for most important use cases (e.g. post handshake client authentication).

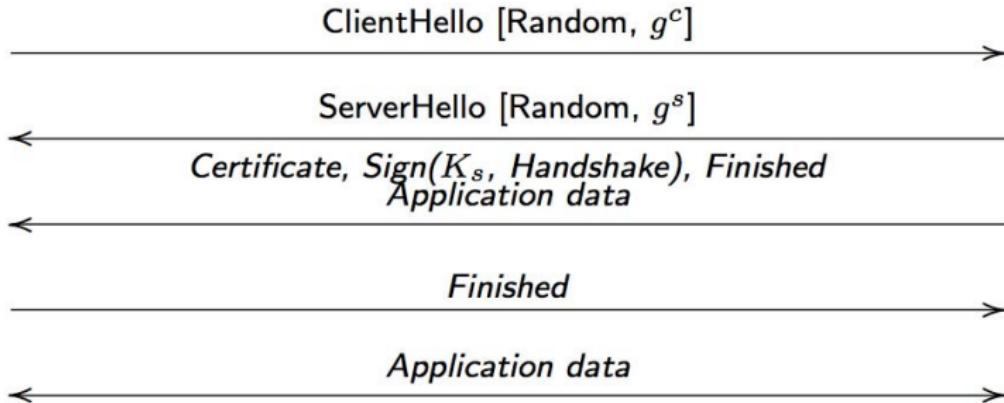
TLS 1.3

- Significant involvement of academic community during the design process.
 - Security analysis of early drafts of the protocol by several teams, using provable security and symbolic analysis.
 - Some significant errors uncovered during development.
 - Analysis on-going: draft spec keeps changing!

TLS 1.3 Handshake – 1-RTT

Client

Server



- Server can send secure data in its first message.
- Client can send secure data in its second message.

TLS 1.3 Handshake – 1-RTT

- Client includes DH share(s) in its first message, along with `ClientHello`, anticipating group that server will prefer.
- Server responds with single DH share in its `ServerHello` response.
- If this works, a forward-secure key is established after 1 round trip (1-RTT).
- Clients can cache groups preferred by popular servers.
- If server does not like DH groups used by client, it sends a `HelloRetryRequest` and a group back to client.
- In this instance, the handshake requires two round trips (2-RTT).

TLS 1.3 Handshake – DH and ECDH groups

- Limited set of DH and ECDH groups will be supported in TLS 1.3.
- Reduces likelihood of fall-back to 2-RTT.
- Removes problem of client not being able to validate groups that was inherent in TLS 1.2 and earlier.
- Removes complexity from implementations.

TLS 1.3 Handshake – DH and ECDH groups

- DH groups:
 - Specified in RFC 7919
 - $|p| = 2048, 3072, 4096, 6144, 8192$.
 - All p are such that $q = (p-1)/2$ is prime.
 - Removes several avenues of attack: backdoored primes, small subgroup attacks, etc (cf. recent work by Fried *et al.*, Valenta *et al.*)
- ECDH groups:
 - Some existing curves from RFC 4492 and 2 new curves in RFC 7748.
 - NIST P256, P384, P521; Curve25519, Curve448.

TLS 1.3 Handshake – o-RTT

- Prior versions of TLS: session resumption feature.
 - Lightweight handshake protocol, exchange of nonces and new key derivation.
 - No public key crypto, but still 1-RTT.
- Under pressure from QUIC design, TLS WG decided to add a **o-RTT** option to TLS 1.3.
 - Enables client to send encrypted data in its first message.
 - Not fully forward secure, since it uses either an old key or a new DH value from client but old DH value from server.
 - After a lot of analysis, it was realised that providing anti-replay for such messages was hard-to-impossible in distributed server environments.

TLS 1.3 Handshake – o-RTT

- **Elegant theoretical solution:** achieve forward secure o-RTT using HIBE + puncturable encryption techniques (e.g., Günther *et al.*, EC'17).
- **Actual solution:** forget about protecting against replay attacks and use the feature only for certain types of data where replay is not an issue.
- Now o-RTT handshakes are bootstrapped using keys from previous protocol runs.
- **Problem:** how to explain security of o-RTT data to developers?
- **Solution:** maintain a separate API for o-RTT data.
- **Residual problem:** performance gain is too tempting for developers to heed warnings about its dangers.

- It was also realised that o-RTT and PSK flows could be unified.
- PSK is an important use case for, e.g. IoT applications.

TLS 1.3 – Other features

- **Post-handshake client authentication:** previously done using renegotiation, now done with special handshake messages.
 - Server sends CertificateRequest message; client responds with Certificate, CertificateVerify, Finished.
- **Key update mechanism:** based on data limits for AES-GCM and ChaCha20Poly1305, derived from security proofs.
- **Record Protocol:** features AEAD only, traffic padding, single plaintext type field and encrypted type, use of masked nonces.
- **Key schedule:** derivations using HKDF and labels; much more complex key schedule; hash for HKDF negotiated in handshake; proper key separation of all keys allowing easier analysis.

Concluding Remarks

Concluding Remarks

- The TLS Handshake Protocol uses mostly “boring” cryptography yet is extraordinarily complex.
 - Much more so than typical key exchange protocols appearing in the scientific literature.
 - Making the protocol resistant to analysis efforts.
- Some protocol design errors were made, but not too many.
- Legacy support for EXPORT cipher suites and long tail of old versions has opened up serious vulnerabilities.
- Lack of formal state-machine description, lack of API specification, and sheer complexity of specifications have led to many serious implementation errors.
- Some, but not all of this, is being fixed in TLS 1.3.

Concluding Remarks

- Public key cryptography has evolved significantly in TLS.
- The largest shift has been from RSA key transport to elliptic curve Diffie-Hellman.
- A second shift now underway is to move to using newer elliptic curves like Curve25519, allowing greater speed and better implementation security.
- A third shift is the move away from SHA-1 in certs.
- A future shift may (will?) be needed to incorporate post-quantum algorithms.
- But implementation vulnerabilities are bound to continue to be discovered.

Elliptic Curves

Why the need for Elliptic Curves?

Discrete logarithm problem example

Let $p =$

10535462803950169753046165829339587319488718149259
13489342608734258717883575185867300386287737705577
93738292587376245199045043066135085968269741025626
82711472830348975632143002371663691740666159071764
72549470083113107138189921280884003892629359

NB: $p = 158(2^{800} + 25) + 1$ and has 807 bits.

Problem:

Find $\lambda \in \mathbb{Z}$ such that

$$2 \equiv 3^\lambda \pmod{p}.$$

Discrete logarithm problem

Let p and L be large primes such that $L|(p - 1)$.

The multiplicative group of integers modulo p contains an element g of order L .

The discrete logarithm problem:

Suppose $h \in \mathbb{Z}_p^*$ also has order L .

Find $\lambda \in \mathbb{Z}$ such that

$$h \equiv g^\lambda \pmod{p}.$$

One-way function:

Fast to compute g^λ but difficult to compute λ .

Generalisation of DLOGs

Can take any (finite) group.

Bad Choices:

- ▶ Additive group \mathbb{Z} or \mathbb{F}_q .
- ▶ Multiplicative group of \mathbb{R} or \mathbb{C} .

Apparently Good Choices:

- ▶ Finite fields \mathbb{F}_q^* .
- ▶ Elliptic curves over finite fields.
- ▶ Ideal class groups of number fields.
- ▶ Jacobian varieties of curves over finite fields.

Subexponential Algorithms

For factoring and the discrete logarithm problem in finite fields \mathbb{F}_q^* there are [index calculus](#) algorithms.

These have subexponential complexity

$$O(\exp(c(\ln N)^{1/3}(\ln \ln N)^{2/3})).$$

For solving the discrete logarithm problem in class groups and Jacobians of curves of sufficiently high genus there are index calculus algorithms of subexponential complexity

$$O(\exp(c(\ln N)^{1/2}(\ln \ln N)^{1/2})).$$

But elliptic curve groups generally have exponential complexity.

Basics on Elliptic Curves

Elliptic Curves

An elliptic curve over a field K is non-singular curve

$$y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

with $a_1, a_2, a_3, a_4, a_6 \in K$.

From these constants we define

$$b_2 = a_1^2 + 4a_2,$$

$$b_4 = a_1 a_3 + 2a_4,$$

$$b_6 = a_3^2 + 4a_6,$$

$$b_8 = a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2,$$

$$c_4 = b_2^2 - 24b_4,$$

$$c_6 = -b_2^3 + 36b_2 b_4 - 216b_6.$$

Elliptic Curves

A curve is called non-singular if it has no singularities.

- ▶ Essentially the “curve” does not cross or intersect itself.

This is easy to detect since the “discriminant” Δ will be zero if the curve is singular

$$\Delta = -b_2^2 b_8 - 8b_4^3 - 27b_6^2 + 9b_2 b_4 b_6.$$

Think of this as related to the discriminant of a polynomial, which is zero when the polynomial has repeated roots.

The curve is considered to be the set of solutions to the equations, plus

- ▶ An additional special point at infinity \mathcal{O}_E .

This is considered to lie infinitely far up the y -axis.

Elliptic Curves

Two curves E and E' are isomorphic over K if there is a bi-rational map between them which preserves the point at infinity.

Two curves with variables X, Y and X', Y' are isomorphic over K if there are constants $r, s, t \in K$ and $u \in K^*$, such that the change of variables

$$X = u^2 X' + r, \quad Y = u^3 Y' + s u^2 X' + t \quad (1)$$

transforms E into E' .

Two most used cases (in classical ECC) are

- ▶ Characteristic p : $K = \mathbb{F}_p$, p a large prime
- ▶ Characteristic 2 : $K = \mathbb{F}_{2^n}$.

In pairing based crypto we also use

- ▶ Characteristic p : $K = \mathbb{F}_{p^n}$ for small n .

Elliptic Curves: Char p

In char p all curves are isomorphic to one of the form

$$E_{A,B} : Y^2 = X^3 + AX + B,$$

in which case we have

$$\Delta = -64A^3 - 432B^2.$$

Two curves in this form $E_{A,B}$ and $E_{A',B'}$ are isomorphic over K if

$$A' = u^4 A \text{ and } B' = u^6 B \text{ for } u \in K^*.$$

Elliptic Curves: Char p

If $-3/A$ is a fourth root in K^* then we can replace $E_{A,B}$ by the curve

$$E_{-3,B'} : Y^2 = X^3 - 3X + B',$$

which will provide a lot of efficiency gains later.

- ▶ In practice it is rare to choose $A \neq -3$ for classical cryptography.

The value $-3/A$ will be a fourth root

- ▶ 25 percent of the time when $p \equiv 1 \pmod{4}$.
- ▶ 50 percent of the time when $p \equiv 3 \pmod{4}$.

Elliptic Curves: Char 2

In char 2 all curves are isomorphic to one of the form

$$E_{A,B} : Y^2 + XY = X^3 + AX^2 + B,$$

where $A \in \{0, \gamma\}$ where γ is a fixed element in B of trace one.

In which case we have

$$\Delta = B.$$

Note for later, the number of elements in $E_{A,B}(K)$ is divisible by 4 if the trace of A is zero, and divisible by 2 otherwise.

Since we want curves with small cofactor and we usually choose fields of odd exponent, e.g. $K = \mathbb{F}_{2^p}$ where p is prime it is common to select

$$A = 1,$$

since this aids efficiency.

The Group Law

Elliptic Curves As Groups

$$E(K) = \{\text{points } (x, y) \in K^2\} \cup \{\mathcal{O}_E\}.$$

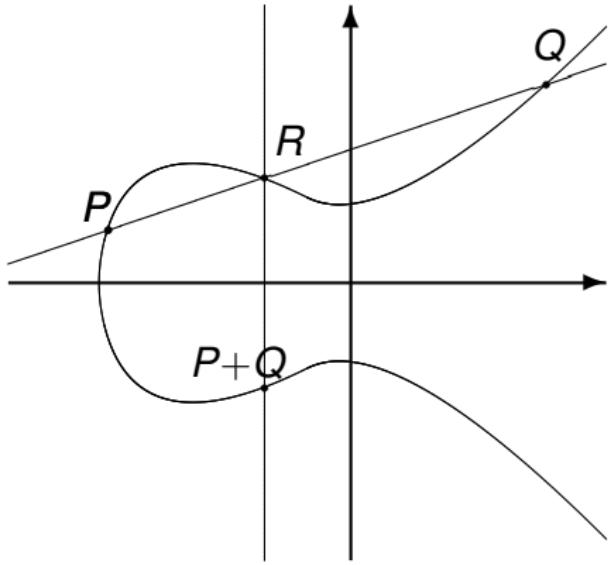
Point Addition

There is a process which, given two points (x_1, y_1) and (x_2, y_2) , gives a third point (x_3, y_3) .

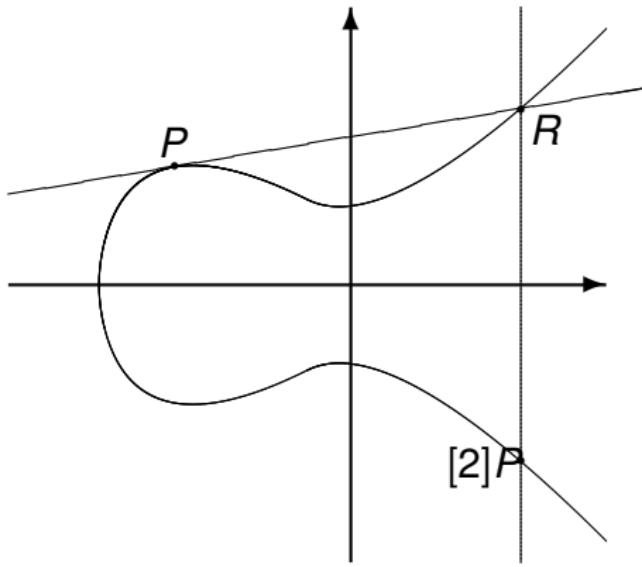
This **addition** process makes the set $E(K)$ an **Abelian group** with identity \mathcal{O}_E .

- ▶ An Abelian group is what you need for a lot of crypto protocols.

Adding two points on an elliptic curve



Doubling a point on an elliptic curve



Addition Formulae

We can write down formulae for the addition law

- ▶ Hence, can compute with the addition law

This can be done with the general equation in any characteristic.

We shall give the simplifications in the two main cases.

Addition Formulae: Char p

Suppose we are in characteristic p

$$E : Y^2 = X^3 + AX + B$$

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on E .

Negation in group law is given by

- ▶ $-P_1 = (x_1, -y_1)$.

Addition Formulae: Char p

Suppose

$$P_3 = (x_3, y_3) = P_1 + P_2$$

then

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2, \\y_3 &= (x_1 - x_3)\lambda - y_1.\end{aligned}$$

where when $x_1 \neq x_2$ we set

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1},$$

and when $x_1 = x_2$ and $y_1 \neq 0$ we set

$$\lambda = \frac{3x_1^2 + A}{2y_1}.$$

Addition Formulae: Char 2

Suppose we are in characteristic 2

$$E : Y^2 + XY = X^3 + AX^2 + B$$

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on E .

Negation in group law is given by

- ▶ $-P_1 = (x_1, y_1 + x_1)$.

Addition Formulae: Char 2

Suppose

$$P_3 = (x_3, y_3) = P_1 + P_2$$

then

$$\begin{aligned}x_3 &= \lambda^2 + \lambda + A + x_1 + x_2, \\y_3 &= (x_1 + x_3)\lambda + x_3 + y_1.\end{aligned}$$

where when $x_1 \neq x_2$ we set

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1},$$

and when $x_1 = x_2 \neq 0$ we set

$$\lambda = \frac{x_1^2 + y_1}{x_1}.$$

Cost of Addition Formulae: Char p

Point Addition:

- ▶ 6 Field Additions (Trivial)
- ▶ 3 General Field Multiplications
- ▶ 1 Field Inversion

Point Doubling:

- ▶ 5 Field Additions (Trivial)
- ▶ 2 Scalar/Field Multiplications (Trivial)
- ▶ 4 General Field Multiplications
- ▶ 1 Field Inversion

Cost of Addition Formulae: Char 2

Point Addition:

- ▶ 9 Field Additions (Trivial)
- ▶ 1 Field squaring (Trivial in Char 2)
- ▶ 2 General Field Multiplications
- ▶ 1 Field Inversion

Point Doubling:

- ▶ 8 Field Additions (Trivial)
- ▶ 2 Field squaring (Trivial in Char 2)
- ▶ 2 General Field Multiplications
- ▶ 1 Field Inversion

Note point doubling requires fewer multiplications than in the char p case.

The ECDLP

Given $P = (x, y)$ and an integer n we can efficiently compute

$$nP = \underbrace{(x, y) + (x, y) + \cdots + (x, y)}_{n \text{ times}}$$

using the **double-and-add** method.

The **Order** of the point P is the smallest number $L > 0$ such that

$$LP = \mathcal{O}_E.$$

Assume that L is a ‘large’ prime.

ECDLP

Suppose that $Q = (x', y')$ is some other point of order L .

Then there is some number λ such that

$$Q = \lambda P.$$

The ECDLP is to find this number λ .

- ▶ This problem is believed to be hard.
- ▶ This gives a one way function.

It is believed in general that the best algorithm to solve this problem takes time

$$O(\sqrt{L}).$$

i.e. fully exponential complexity.

Choices of Finite Field Arithmetic

Recap on Finite Fields

A **Field** is a set with two operations $(G, \times, +)$ such that

- ▶ $(G, +)$ is an abelian group, identity denoted by 0.
- ▶ $(G \setminus \{0\}, \times)$ is an abelian group
- ▶ $(G, \times, +)$ satisfies the **distributive law**

Distributive law

For all $f, g, h \in (G, \times, +)$

$$f \times (g + h) = (f \times g) + (f \times h).$$

Examples

Rational numbers, real numbers, complex numbers, integers modulo p .

Fields

We define the set of invertible elements of $\mathbb{Z}/N\mathbb{Z}$ as

$$(\mathbb{Z}/N\mathbb{Z})^* = \{a \in \mathbb{Z}/N\mathbb{Z} : \gcd(a, N) = 1\}.$$

The set $(\mathbb{Z}/N\mathbb{Z})^*$ is always a group with respect to multiplication and clearly has size $\phi(N)$.

When N is a prime p we have

$$\mathbb{Z}/N\mathbb{Z}^* = \{1, \dots, p - 1\}.$$

We define the sets

$$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \{0, \dots, p - 1\} \quad \text{and} \quad \mathbb{F}_p^* = (\mathbb{Z}/p\mathbb{Z})^* = \{1, \dots, p - 1\}.$$

We call \mathbb{F}_p a **finite field of characteristic p** .

Finite fields are of central importance in **coding theory** and **cryptography**.

Finite Field Arithmetic

It is crucial to have a good finite field arithmetic.

A number of different choices have been proposed.

We shall now recap on the main two choices.

- ▶ \mathbb{F}_{2^p} , p prime
- ▶ \mathbb{F}_p , p prime

Characteristic Two Fields

Characteristic Two Fields

Of particular interest are fields of char 2.

Take an **irreducible** binary polynomial f of degree n and let \mathbb{F}_{2^n} denote all the binary polynomials of degree $< n$.

Addition in \mathbb{F}_{2^n} is defined as

- ▶ $a \oplus b = a + b \pmod{2}$
- ▶ Note this means $-a = a$.

Multiplication in \mathbb{F}_{2^n} is defined as

- ▶ $a \otimes b = a \cdot g \pmod{f}$.
- ▶ Inversion is performed by a variant of the Euclidean algorithm for polynomials.

Characteristic Two Fields

Often write

$$\mathbb{F}_{2^n} = \mathbb{F}_2[x]/f$$

to denote working modulo f .

Set of non-zero elements denoted by $\mathbb{F}_{2^n}^*$

- ▶ This is the **multiplicative subgroup** of the field

Char 2 Example

Let $f = x^6 + x + 1$ (this is **irreducible**) The finite field of 2^6 elements can then be identified with

- ▶ Bit strings of length six bits
- ▶ Binary polynomials of degree less than or equal to five

$$a = 001101 = x^3 + x^2 + 1$$

$$b = 101011 = x^5 + x^3 + x + 1$$

$$a \oplus b = 100110 = x^5 + x^2 + x$$

- ▶ Since the two x^3 and the two 1 terms cancel, as we are working mod two.
- ▶ Notice, we are simply taking the exclusive-or of the bit string representation.

Char 2 Example

Recap $f = x^6 + x + 1$, $a = 001101 = x^3 + x^2 + 1$,
 $b = 101011 = x^5 + x^3 + x + 1$.

Since f is sparse reduction mod f done using rewriting, as
 $x^6 = x + 1 \pmod{f}$,

$$\begin{aligned} a \otimes b &= (x^3 + x^2 + 1) \cdot (x^5 + x^3 + x + 1) \\ &= x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + x + 1 \\ &= x^6 \cdot (x^2 + x + 1) + x^4 + x^3 + x^2 + x + 1 \\ &= (x + 1) \cdot (x^2 + x + 1) + x^4 + x^3 + x^2 + x + 1 \\ &= (x^3 + 1) + (x^4 + x^3 + x^2 + x + 1) \\ &= x^4 + x^2 + x. \end{aligned}$$

i.e. $a \otimes b = 010110 = x^4 + x^2 + x$.

Char 2 Example

Since f is assumed irreducible, every polynomial $a \neq 0$ is coprime to f .

Hence, using a binary polynomial version of the extended GCD algorithm we can find u and v so that

$$u \cdot a + v \cdot f = 1 \pmod{2}.$$

In which case $a^{-1} = u$ in \mathbb{F}_{2^n} .

If $a = x^3 + x^2 + 1$ and $f = x^6 + x + 1$ then taking $u = x^5 + x^3$ and $v = x^2 + x + 1$ gives us

- ▶ $u \cdot a + v \cdot f = 1 \pmod{2}$

and so

- ▶ $a^{-1} = u = x^5 + x^3 = 101000$.

Choice of Defining Polynomial

All char 2 fields of the same degree n are **isomorphic**.

- ▶ This means it does not depend on which polynomial f we take.
- ▶ Different f 's give different representations of the same thing.

Let $f(x)$ and $g(y)$ be irreducible polynomials of degree n . Then there are polynomial's $r(x)$ and $s(y)$ such that one can map one field into the other via

- ▶ $x \pmod{f(x)} \longrightarrow s(y) \pmod{g(y)}$
- ▶ $y \pmod{g(y)} \longrightarrow r(x) \pmod{f(x)}$

This means we can select the best irreducible polynomial f for our own implementation.

- ▶ Requires the mapping $s(y)$ only when talking to someone else's implementation which uses $g(y)$ instead.

Characteristic Two : Composite Extension

These are fields of the form $\mathbb{F}_{2^{n \cdot m}}$

For a while some people proposed these (of course backed up by patents).

- ▶ The IETF standards include such finite fields.
- ▶ They provide a number of performance advantages

Problem is that such fields when used in ECC are susceptible to **Weil Descent** attacks.

- ▶ Whilst such attacks are often not practical they cast sufficient concern to mean we no longer use such fields.

Characteristic Two : Prime Extension

$K = \mathbb{F}_{2^p}$ With p prime, eg $p = 163, 191$.

- ▶ Trinomial Bases
- ▶ Pentanomial Bases
- ▶ Normal Bases

All are very good in hardware, normal bases are very good.

All three occur in standards documents ANSI/NIST etc.

- ▶ These days Normal Bases less used.

Large Prime Characteristic Fields

Large Prime Characteristic

$$K = \mathbb{F}_p$$

Can be implemented in a number of ways.

- ▶ Montgomery arithmetic (general prime)
- ▶ Barrett Reduction (general prime)
- ▶ Generalised Mersenne Primes (special prime)

Most popular method for a general modulus is Montgomery arithmetic.

In deployed classical ECC systems the most popular choice are Generalised Mersenne Primes.

GM Primes

GM Primes

$$K = \mathbb{F}_p$$

Standards bodies have settled on GM-Primes as the main recommend fields.

- ▶ GM-Primes give significant performance advantages.
- ▶ Their special form means one has **significant** performance improvements.
- ▶ Montgomery Mult takes about $2n(n + 1)$ word multiplications, whereas for GM-Primes this is only n^2 .

A GM-prime is one of the form

$$p = f(2^{32}) \text{ or } f(2^{64}),$$

where f is a "low weight" polynomial.

- ▶ Eg. $p = 2^{192} - 2^{64} - 1$ is popular.

GM Primes

Suppose we take $p = 2^{192} - 2^{64} - 1$ as an example and we want to compute

$$z = x \cdot y \pmod{p}.$$

To perform modular multiplication we first do a standard school book (or Karatsuba) multiplication

$$a = x \cdot y = a_1 2^{192} + a_0$$

where $a_0, a_1 < 2^{192}$.

- ▶ This requires at most 36 32-bit word multiplications plus some additions

GM Primes

We now need to produce $z = a \pmod{p}$, but note that mod p we have

$$2^{192} = 2^{64} + 1$$

and so

$$\begin{aligned} a &\equiv a_1(2^{64} + 1) + a_0 \pmod{p} \\ &= b_1 2^{192} + b_0, \end{aligned}$$

where $b_0 < 2^{192}$ but $b_1 < 2^{65}$.

We then repeat to obtain

$$\begin{aligned} b &\equiv b_1(2^{64} + 1) + b_0 \pmod{p} \\ &= z. \end{aligned}$$

GM Primes

Notice the reduction stage just involved some shifting and addition

- ▶ i.e. very cheap operations.

This total time is dominated by the 36 32-bit word multiplications.

If we did Montgomery arithmetic on similar size numbers we would require

$$2 \cdot 6 \cdot (6 + 1) = 84$$

32-bit word multiplications.

Hence the GM-Prime version will be around twice as fast.

OEF Fields

OEF Fields

$$K = \mathbb{F}_{p^n}$$

These have appeared in a number of papers for classical ECC.

OEF Fields.

- ▶ OEF fields choose p close to the word size.
- ▶ The equation defining K over \mathbb{F}_p is chosen to be very simple,

$$x^n - 2.$$

OEF Fields give very good implementations.

- ▶ Very fast field inversion.

Mainly utilized in pairing based systems, for the second field

Curve Arithmetic

Curve Arithmetic

Points can be added/doubled in a number of formats:

Affine:

- ▶ $P = (X, Y)$.

Projective (Standard):

- ▶ $P = (X, Y) = (x/z, y/z)$.

Projective (Jacobian):

- ▶ $P = (X, Y) = (x/z^2, y/z^3) = (x, y, z)$.

Chudnovsky:

- ▶ As Jacobian but store $P = (x, y, z, z^2, z^3)$.

Lopez-Dahab:

- ▶ $P = (X, Y) = (x/z, y/z^2) = (x, y, z)$

Mixed:

- ▶ A combination of any of the above.

Curve Arithmetic

Standard projective coordinates are rarely used.

We will concentrate on

- ▶ Affine,
- ▶ Jacobean Projective/Lopez-Dahab
- ▶ A mixture of the two.

Main problem with Affine is that we require field inversions.

- ▶ Field inversions are generally much slower than multiplications.

Projective coordinates allow us to trade a number of multiplications for an inversion.

We shall see later that it is point doubling which is most important.

Cost of Curve Arithmetic: Char p

In odd characteristic it is often best to use Jacobean Projective coordinates:

Operation	Affine	Projective	Mixed
Addition	$3M + 1I$	$16M$	$11M$
Doubling	$4M + 1I$	$10M$	n/a

In next few slides we give the formulae for computing

- ▶ $P_3 = P_1 + P_2$

with

- ▶ $P_i = (X_i, Y_i, Z_i)$

when the curve is given by

$$Y^2 = X^3 + AX + B.$$

Projective Addition Formulae: Char p

$$\begin{aligned}\lambda_1 &= X_1 Z_2^2 & 2M \\ \lambda_2 &= X_2 Z_1^2 & 2M \\ \lambda_3 &= \lambda_1 - \lambda_2 \\ \lambda_4 &= Y_1 Z_2^3 & 2M \\ \lambda_5 &= Y_2 Z_1^3 & 2M \\ \lambda_6 &= \lambda_4 - \lambda_5 \\ \lambda_7 &= \lambda_1 + \lambda_2 \\ \lambda_8 &= \lambda_4 + \lambda_5 \\ Z_3 &= Z_1 Z_2 \lambda_3 & 2M \\ X_3 &= \lambda_6^2 - \lambda_7 \lambda_3^2 & 3M \\ \lambda_9 &= \lambda_7 \lambda_3^2 - 2X_3 \\ Y_3 &= (\lambda_9 \lambda_6 - \lambda_8 \lambda_3^3)/2 & 3M \\ && \hline && 16M\end{aligned}$$

Projective Addition Formulae: Char p

In previous slide

$$\lambda_3 = 0 \text{ if and only if } P_1 = \pm P_2$$

$$\lambda_3 = \lambda_6 = 0 \text{ if and only if } P_1 = P_2$$

- ▶ In this case need to execute a point doubling (see later)

A mixed addition is when $Z_1 = 1$ in which case we simplify the formulae as in the next slide

Projective (Mixed) Addition Formulae: Char p

$$\begin{aligned}\lambda_1 &= X_1 Z_2^2 & 2M \\ \lambda_3 &= \lambda_1 - X_2 \\ \lambda_4 &= Y_1 Z_2^3 & 2M \\ \lambda_6 &= \lambda_4 - Y_2 \\ \lambda_7 &= \lambda_1 + X_2 \\ \lambda_8 &= \lambda_4 + Y_2 \\ Z_3 &= Z_2 \lambda_3 & 1M \\ X_3 &= \lambda_6^2 - \lambda_7 \lambda_3^2 & 3M \\ \lambda_9 &= \lambda_7 \lambda_3^2 - 2X_3 \\ Y_3 &= (\lambda_9 \lambda_6 - \lambda_8 \lambda_3^3)/2 & 3M \\ && \hline && 11M\end{aligned}$$

Projective Doubling Addition Formulae: Char p

A point doubling is performed via

$$\begin{array}{rcl} \lambda_1 & = & 3X_1^2 + AZ_1^4 & 4M \\ Z_3 & = & 2Y_1Z_1 & 1M \\ \lambda_2 & = & 4X_1 Y_1^2 & 2M \\ X_3 & = & \lambda_1^2 - 2\lambda_2 & 1M \\ \lambda_3 & = & 8Y_1^4 & 1M \\ Y_3 & = & \lambda_1(\lambda_2 - X_3) - \lambda_3 & 1M \\ & & & \hline & & & 10M \end{array}$$

Projective Doubling Addition Formulae: Char p

Recall we said usually $A = -3$ then

$$\begin{array}{rcl} \lambda_1 & = & 3X_1^2 + AZ_1^4 = 3(X_1 + Z_1^2)(X_1 - Z_1^2) & 2M \\ Z_3 & = & 2Y_1Z_1 & 1M \\ \lambda_2 & = & 4X_1 Y_1^2 & 2M \\ X_3 & = & \lambda_1^2 - 2\lambda_2 & 1M \\ \lambda_3 & = & 8Y_1^4 & 1M \\ Y_3 & = & \lambda_1(\lambda_2 - X_3) - \lambda_3 & 1M \\ \hline & & & 8M \end{array}$$

Cost of Curve Arithmetic: Char 2

In even characteristic it is often best to use Lopez-Dahab coordinates:

Operation	Affine	Lopez-Dahab	Mixed
Addition	$2M + 1I$	13 M	8 M
Doubling	$2M + 1I$	4 M	n/a

Note squaring is free in even characteristic.

In next few slides we give the formulae for computing

$$\triangleright P_3 = P_1 + P_2$$

with

$$\triangleright P_i = (X_i, Y_i, Z_i)$$

when the curve is given by

$$Y^2 + XY = X^3 + AX^2 + B.$$

Projective Addition Formulae: Char 2

λ_1	$= X_1 Z_2$	$1M$
λ_2	$= X_2 Z_1$	$1M$
λ_3	$= \lambda_1 + \lambda_2$	
λ_4	$= \lambda_1^2$	free
λ_5	$= \lambda_2^2$	free
λ_6	$= \lambda_4 + \lambda_5$	
λ_7	$= Y_1 Z_2^2$	$1M$
λ_8	$= Y_2 Z_1^2$	$1M$
λ_9	$= \lambda_7 + \lambda_8$	
λ_{10}	$= \lambda_3 \lambda_9$	$1M$
Z_3	$= Z_1 Z_2 \lambda_6$	$2M$
X_3	$= \lambda_1(\lambda_8 + \lambda_5) + \lambda_2(\lambda_7 + \lambda_4)$	$2M$
Y_3	$= (\lambda_1 \lambda_{10} + \lambda_7 \lambda_6) \lambda_6 + (\lambda_{10} + Z_3) X_3$	$\frac{4M}{13M}$

Projective (Mixed) Addition Formulae: Char 2

$$\begin{array}{lll} \lambda_1 & = Z_2 Y_1 + Y_2 & 1M \\ \lambda_2 & = Z_2 X_1 + X_2 & 1M \\ \lambda_3 & = Z_2 \lambda_2 & 1M \\ Z_3 & = \lambda_3^2 & \text{free} \\ \lambda_5 & = Z_3 X_1 & 1M \\ \lambda_6 & = X_1 + Y_1 \\ X_3 & = \lambda_1^2 + \lambda_3(\lambda_1 + \lambda_2^2 + A\lambda_3) & 2M \\ Y_3 & = (\lambda_5 + X_3)(\lambda_3\lambda_1 + Z_3) + \lambda_6 Z_3^2 & 3M \\ & & \hline & & 9M \end{array}$$

If $A = 1$ then this reduces to $8M$.

Projective Doubling Addition Formulae: Char 2

A point doubling is performed via

$$\begin{array}{lll} \lambda_1 & = & X_1^2 \\ \lambda_2 & = & \lambda_1 + Y_1 \\ \lambda_3 & = & X_1 Z_1 \\ Z_3 & = & \lambda_3^2 \\ \lambda_5 & = & \lambda_2 Z_3 \\ X_3 & = & \lambda_2^2 + \lambda_3 + A Z_3 \\ Y_3 & = & (Z_3 + \lambda_5) X_3 + \lambda_1^2 Z_3 \end{array} \quad \begin{array}{l} \text{free} \\ \\ \\ 1M \\ \text{free} \\ 1M \\ 1M \\ \hline 5M \end{array}$$

Which reduces to $4M$ if we choose $A = 1$.

Projective Formulae Summary

Odd Characteristic

Operation	Affine	Projective	Mixed
Addition	$3M + 1I$	$16M$	$11M$
Doubling	$4M + 1I$	$10M$	n/a

Even Characteristic

Operation	Affine	Lopez-Daheb	Mixed
Addition	$2M + 1I$	$13 M$	$8 M$
Doubling	$2M + 1I$	$4 M$	n/a

Doubling is **very** fast in even characteristic.

- ▶ This can make up for the slow software field arithmetic in a final implementation.

Point Multiplication

Point Multiplication

The basic cryptographic operation is to compute

$$Q = [d]P$$

for some integer d and point P .

The binary method:

- ▶ $Q = 0$.
- ▶ For $j = \log_2(d) - 1$ to 0
 - ▶ $Q = [2]Q$.
 - ▶ If $d_j = 1$ then $Q = Q + P$.

This requires

- ▶ Exactly $\log_2(d)$ point doublings.
- ▶ About $\log_2(d)/2$ general point additions.
 - ▶ If P affine and Q projective used mixed addition

We can reduce the number of general point additions.

Point Multiplication

m-ary method

The binary method uses a fixed window of size 2.

The *m*-ary method uses a fixed window of size $m = 2^r$.

- ▶ Taking m bits at a time.

Requires precomputation of

$$2^{r-1} - 1$$

general point additions.

Then requires

- ▶ $\log_2(d)$ doublings.
- ▶ About $\log_2(d)/r$ general point additions.

Point Multiplication

Sliding Window Method

This method slides the window of length m across runs of zero's in the binary expansion.

Requires precomputation of

$$2^{r-1} - 1$$

general point additions.

Then requires

- ▶ $\log_2(d)$ doublings.
- ▶ About $\log_2(d)/(r + 1)$ general point additions.

Point Multiplication

Signed Window Methods

For elliptic curves negation comes for free

$$-P = (x, -y) \text{ or } (x, y + x).$$

Hence we could used a signed binary representation.

$$7 = 2^2 + 2 + 1 \text{ or } 7 = 2^3 - 1.$$

This allows us to reduce the number of point additions even further

- ▶ In both the m -ary and the sliding window algorithms.

Signed Sliding Window Method

The following precomputation is done once for each point P :

Precomputation

- ▶ $P_1 = P, P_2 = [2]P.$
- ▶ For $i = 1$ to $2^{r-2} - 1$
 - ▶ $P_{2i+1} = P_{2i-1} + P_2..$
- ▶ $Q = P_{d_{l-1}}.$

These are computed in affine coordinates

Next we encode the number d Set

$$d = \sum_{i=0}^{l-1} d_i 2^{e_i}$$

with $e_{i+1} - e_i \geq r$ and

$$d_i \in \{\pm 1, \pm 3, \dots, \pm 2^{r-1} - 1\}.$$

Signed Sliding Window Method

Main Loop

- ▶ For $i = l - 2$ to 0
 - ▶ $Q = [2^{e_{i+1} - e_i}]Q.$
 - ▶ If $l_i > 0$ then $Q = Q + P_{d_i}.$
 - ▶ Else $Q = Q - P_{-d_i}.$
- ▶ $Q = [2^{e_0}]Q.$

The Q is held in projective coordinates

- ▶ Since P_i are affine can use mixed addition
- ▶ Can use efficient projective doubling formulae

Point Multiplication

The signed sliding window method generally comes out to be the fastest.

Remember

In all cases CPU time is dominated by the time to compute

$$\log_2(d)$$

point doublings.

Lesson

Optimise the doubling operation at all times.

- ▶ This is often ignored.

Point Multiplication

Notice that doubling in char 2 requires less multiplications than in char p .

This can often lead (depending on the processor or the implementation) that the relative advantage of char p field multiplication can be cancelled out by the doubling operation.

This will become even more pronounced as the new instruction set extensions to Intel and other chips become more prevalent, since these will provide native carry-free multipliers.

- ▶ i.e. char 2 field multiplications will run as fast as char p field multiplications
- ▶ Indeed possibly faster.

Key Agreement

Elliptic Curve Diffie-Hellman

Easiest to understand of all the protocols.

Two people, Alice and Bob, want to agree a shared secret.

$E(\mathbb{F}_q)$ is an elliptic curve over a finite field for which ECDLP is hard.

P is a point of large prime order.

EC-DH

$$\begin{array}{ccc} \text{Alice} & & \text{Bob} \\ x & \xrightarrow{[x]P} & [x]P \\ [y]P & \xleftarrow{[y]P} & y \end{array}$$

Alice can now compute

$$K_A = [x]([y]P) = [xy]P$$

Bob can now compute

$$K_B = [y]([x]P) = [xy]P$$

and

$$K_A = K_B$$

Very small bandwidth if one uses point compression.

EC-DHP

Given

$$[x]P \text{ and } [y]P$$

the problem of recovering

$$[xy]P$$

is called the [Elliptic Curve Diffie-Hellman Problem](#) (ECDHP).

If we can solve ECDLP then we can solve ECDHP.

It is unknown if the other implication holds.

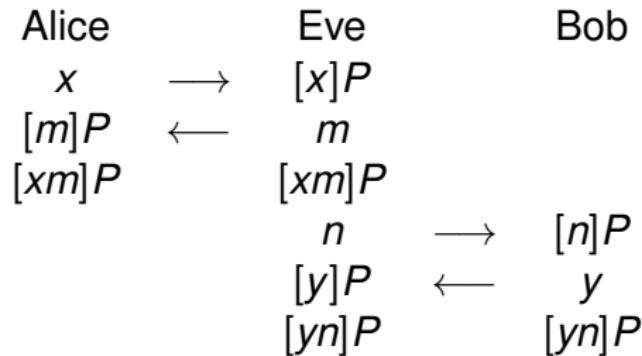
A proof of equivalence of the DHP and DLP for many black box groups follows from work of Boneh, Maurer and Wolf.

This proof uses elliptic curves in a crucial way.

- ▶ EC-DH is standardised in ANSI X9.63.

EC-DH

In practice life is not so simple, for example in EC-DH we have a man-in-the-middle attack



Alice agrees a key with Eve, thinking it is Bob Bob agrees a key with Eve, thinking it is Alice

Eve can now examine communications as they pass through her (she acts as a router).

Diffie-Hellman

Diffie-Hellman on its own is not enough.

For example how does Alice know who she is agreeing a key with ?

- ▶ Is it Bob or Eve ?

One way around is for

- ▶ Alice to **sign** her message to Bob
- ▶ Bob to **sign** his message to Alice.

In that way both parties know who they are talking to.

Signed Diffie-Hellman

Assuming we can construct secure signature schemes we have now solved the key distribution problem:

Authentic public keys are obtained from a CA.

Then secure session keys are obtained using signed Diffie-Hellman

$$\begin{array}{ccc} \text{Alice} & & \text{Bob} \\ ([a]P, \text{Sign}_{\text{Alice}}([a]P)) & \xrightarrow{\hspace{2cm}} & \\ & \xleftarrow{\hspace{2cm}} & ([b]P, \text{Sign}_{\text{Bob}}([b]P)) \end{array}$$

However, it is more common to use the STS-protocol in this situation

STS Key Agreement Protocol

When using signed Diffie–Hellman it is common to adopt the **Station-to-Station** protocol, or STS protocol

- ▶ $A \rightarrow B : [x]P$
- ▶ $B \rightarrow A : [y]P, \{ \text{Sig}_B([y]P, [x]P) \}_{K_{ab}}$
- ▶ $A \rightarrow B : \{ \text{Sig}_A([x]P, [y]P) \}_{K_{ab}}$

where $K_{ab} = g^{xy}$

STS provides forward secrecy, but has some subtle problems

Has **unknown key share attack**

- ▶ This attack requires adversary to obtain “invalid” certificates from a CA
- ▶ Very damaging if **duplicate signatures** can be found

MQV Protocol

System setup

Alice and Bob generate a public/private key pair each

$$(A = [a]P, a) \text{ and } (B = [b]P, b).$$

Via some means (eg a certificate)

- ▶ Bob knows A is authentic
- ▶ Alice knows B is authentic

They now want to agree on a secret session key to which they both contribute a random nonce

- ▶ nonce = number which is used once and then thrown away

Use of the nonce's provides them with forward secrecy

MQV Authenticated Key Exchange

This is the most efficient deployed authenticated key exchange protocol

- ▶ Most analysed authenticated key exchange mechanism available
- ▶ NSA like this one

The key exchange Alice and Bob now generate a public/private ephemeral key pair each

$$(\textcolor{blue}{C} = [\textcolor{red}{c}]P, \textcolor{red}{c}) \text{ and } (\textcolor{blue}{D} = [\textcolor{red}{d}]P, \textcolor{red}{d}).$$

They exchange $\textcolor{blue}{C}$ and $\textcolor{blue}{D}$.

Hence to some extent this looks like a standard Diffie-Hellman exchange with no signing.

However the final session key will also depend on $\textcolor{blue}{A}$ and $\textcolor{blue}{B}$.

Determining the Session Key

Assume you are Alice

- ▶ You know A, B, C, D, a and c

Let l denote half the bit size of the group G

- ▶ e.g. $l = 160/2 = 80$

Shared secret K computed via

- ▶ Convert C to an integer i
- ▶ Put $s = (i \pmod{2^l}) + 2^l$
- ▶ Convert D to an integer j
- ▶ Put $t = (j \pmod{2^l}) + 2^l$
- ▶ Put $h = c + sa$.
- ▶ Put $K = [h]([t](DB))$

Why does MQV this work ?

Note that s and t seen by Alice, are swapped when seen by Bob,

- ▶ $s_{Alice} = t_{Bob}$
- ▶ $t_{Alice} = s_{Bob}$

Let

- ▶ h_{Alice} denote the h seen by Alice
- ▶ h_{Bob} denote the h seen by Bob

Then

- ▶ $P = g^{h_{Alice} \cdot h_{Bob}}$

You should check this for yourself

MQV however still suffers from an unknown key-share attack

- ▶ But a very subtle attack, hence probably not important in practice

Encryption

Hybrid Encryption : KEMs and DEMs

Before introducing ECIES we recap on the modern method for creating public key encryption schemes.

Most public key schemes are used in a hybrid manner.

- ▶ A public key system is used to encrypt a symmetric key (**key encapsulation mechanism**).
- ▶ The symmetric key is used to encrypt the actual data (**data encapsulation mechanism**).

This is now formalised: the **KEM-DEM** or **hybrid encryption** methodology.

Key Encapsulation Mechanisms

Key Encapsulation Mechanism

A KEM is an algorithm which takes as input a public key y and outputs a pair (K, C) where

- ▶ K is a key for a symmetric encryption function (e.g. DES or AES) and
- ▶ C is an encapsulation (encryption) of K using y .

The inverse, decapsulation algorithm takes as input (C, x) where

- ▶ C is an encapsulation under y of some key K - or it should be! - and
- ▶ x is the private key corresponding to y .

It outputs

- ▶ either \perp if C is an invalid encapsulation, or
- ▶ K if C is an encapsulation of the key K .

Data Encapsulation Mechanisms

Data Encapsulation Mechanism

A DEM is a symmetric algorithm which on input of

- ▶ a message M and
- ▶ a symmetric key K

outputs an encryption E of M using key K .

The inverse operation takes as input

- ▶ (E, K)

and outputs either

- ▶ \perp if E is an invalid ciphertext or
- ▶ M if E is an encryption of M under the key K .

A KEM-DEM Hybrid Cipher

Using the primitives we have been discussing, a KEM-DEM hybrid encryption scheme can then be created as follows.

Encryption

- ▶ $(K, C) \leftarrow \text{KEM}(y)$
- ▶ $E \leftarrow \text{DEM}(M, K)$
- ▶ Return (C, E)

Decryption

- ▶ $K \leftarrow \text{KEM}^{-1}(C, x)$
- ▶ If $K = \perp$ then return \perp
- ▶ $M \leftarrow \text{DEM}^{-1}(E, K)$
- ▶ If $M = \perp$ then return \perp
- ▶ Return M

DEM : Construction

To construct a secure DEM we take

- ▶ a **secure** (under passive attack) block cipher E and
- ▶ a **secure** (cannot produce a MAC without the corresponding key) MAC function MAC .

The function $\text{DEM}(M, K)$ is then constructed as follows.

- ▶ Split K into k_0 and k_1 .
- ▶ $c_0 \leftarrow E(M, k_0)$.
- ▶ $c_1 \leftarrow \text{MAC}(c_0, k_1)$.
- ▶ Return $C = (c_0, c_1)$.

KEM-DEM Security Properties

It can be shown that if

- ▶ a KEM is secure

and

- ▶ a DEM is secure

then the combined KEM-DEM hybrid scheme is IND-CCA2 secure.

Thus the KEM-DEM approach to public key encryption allows us to build schemes in a **modular** fashion:

- ▶ design a secure KEM;
- ▶ design a secure DEM;
- ▶ combine them to obtain a secure encryption scheme.

Each component can be designed independently.

This is the standard ECC encryption algorithm.

- ▶ Based on Abdalla, Bellare and Rogaway's DHAES protocol
- ▶ Secure under a non-standard assumption (Abdalla et. al.)
- ▶ Secure under in the ROM (Abdalla et. al.)
- ▶ Secure under in the Generic Group Model (Smart)

IES stands for integrated encryption scheme

- ▶ The scheme works like static Diffie-Hellman followed by symmetric encryption.
- ▶ We use Diffie-Hellman as a KEM
- ▶ Then encrypt the message with a DEM

EC-IES Components

A symmetric encryption scheme $\text{SYM} = (E_k, D_k)$,

- ▶ Key space K_1

A MAC function MAC_k

- ▶ Key space K_2 .

A key derivation function V .

- ▶ The key derivation function V will map group elements
- ▶ to the key space of both the encryption and MAC functions.

The scheme ECIES is defined as a triple of randomised algorithms,

- ▶ **{keygen, enc, dec}**.

EC-IES KeyGen

- ▶ $d \leftarrow \{1, \dots, q\}.$
- ▶ $Q \leftarrow [d]P.$
- ▶ Return $(Q, d).$

EC-IES Encryption

- ▶ $k \leftarrow \{1, \dots, q\}.$
- ▶ $U \leftarrow [k]P.$
- ▶ $T \leftarrow [k]Q.$
- ▶ $(k_1, k_2) \leftarrow V(T)$
- ▶ $c \leftarrow E_{k_1}(m)$
- ▶ $r \leftarrow \text{MAC}_{k_2}(c)$
- ▶ Return $e \leftarrow U \| c \| r.$

The cipher text is $(U, c, r).$

- ▶ U is needed to agree a key
- ▶ c is the actual encrypted message
- ▶ r is used to avoid adaptive chosen ciphertext attacks

The data item U can be compressed to reduce bandwidth.

EC-IES Decryption

- ▶ Parse e as $U\|c\|r$
- ▶ $T \leftarrow [d]U$
- ▶ $(k_1, k_2) \leftarrow V(T)$
- ▶ If $r \neq \text{MAC}_{k_2}(c)$
 - ▶ Return **Invalid**
- ▶ $m \leftarrow D_{k_1}(c).$
- ▶ Return $m.$

ECIES

EC-IES makes it easy to encrypt long messages

Standardized in a number of places

- ▶ ANSI X9.63
- ▶ IEEE P1363
- ▶ SEC 1
- ▶ etc

Signatures

EC-DSA

Variant of the American DSA algorithm as specified in NIST FIPS 186.

FIPS 186.2 contains the new version including EC-DSA,

- ▶ This is also in ANSI X9.62, IEEE P1363 and SECG

With all digital signature algorithms one actually signs a **hash** of the message, M .

- ▶ For ECC this hash function is always $SHA - 1/SHA - 2$.
- ▶ $SHA - 1$ (resp. $SHA - 2$) takes an arbitrary length input and produces a 160 (resp 256 etc) bit output.
- ▶ We interpret this output bit string as a number.

We also interpret the x -coordinate of a point as a number.

- ▶ Even when $K = \mathbb{F}_{2^p}$.

EC-DSA : System set up

E an elliptic curve over $K = \mathbb{F}_{p^n}$.

P a point of large prime order, q .

$$\#E(K) = hq$$

h is called the cofactor.

The set $\{K, E, q, h, P\}$ is called the domain parameters.

Private Key : $d \in_R [1, \dots, q - 1]$. Public Key : $Q = [d]P$.

EC-DSA : Signing

Choose $k \in_R [1, \dots, q - 1]$.

Compute $[k]P = (x, y)$.

Convert x to an integer, $r, \text{ mod } q$.

If $r \equiv 0$ then goto beginning.

Put $e = \text{SHA-1}(M)$.

Compute

$$s \equiv (e + dr)/k \pmod{q}.$$

If $s \equiv 0$ then goto beginning.

Return (r, s) as the signature.

EC-DSA : Verifying

Put $e = \text{SHA-1}(M)$.

Reject if $r, s \notin [1, \dots, q - 1]$.

Compute

$$u_1 \equiv e/s \pmod{q}.$$

$$u_2 \equiv r/s \pmod{q}.$$

Set $R = (x, y) = [u_1]P + [u_2]Q$.

Reject if R is at infinity.

Convert x to an integer, l , modulo q .

Accept if and only if $r \equiv l$.

EC-DSA

Need to make sure ephemeral exponent is truly random.

Signing is much faster than RSA

- ▶ Verification is slower.

Size of signature is much smaller than RSA.

Scales much better than RSA or DSA over time.

Schnorr Signatures

An important DLP based signature scheme is that of Schnorr.

- ▶ It occurs in many ZK proofs and as parts of other protocols.
- ▶ It is the simplest among the DLP based schemes that are **provably secure**.
- ▶ The signing and verifying operations are simpler than those for DSA.
- ▶ Is the basis for many other protocols.
- ▶ Standardisation by ISO

Each user generates a secret signing key x at random and such that

- ▶ $0 < x < q$.

Public key is $Q = xP$.

Schnorr Signatures : Signing

To sign a message M the signer proceeds as follows.

- ▶ Signer chooses a random **ephemeral key**: $0 < k < q$.
- ▶ Signer computes $R = kP$.
- ▶ Signer computes one-way hash $m = H(R||M)$.
- ▶ Finally, signer computes

$$s = (k + mx) \pmod{q}.$$

The signature on M is the pair (m, s) .

Schnorr Signatures : Verification

To verify a signature (m, s) on a message M under public key Q , the verifier proceeds as follows.

The verifier computes

$$R' = sP - mQ.$$

If the signature is valid we have

$$R' = (k + mx)P - xmP = kP.$$

So, the verifier accepts signature if and only if

$$m = H(R'||M).$$

ECDLP

How hard is the ECDLP?

Black box group means there is no information about the representation.

- ▶ Only generic algorithms are available.

Theorem: (V. Shoup)

In a black box group of prime order L it takes at least $O(\sqrt{L})$ operations to solve the discrete logarithm problem.

Compare to factoring where underlying problem is sub-exponential

Generic Algorithms

Baby step giant step (Shanks)

Want to find $0 \leq \lambda < L$ such that $Q = \lambda P$ in $E(\mathbb{F}_q)$.

Put $M = \lceil \sqrt{L} \rceil$ (or $M = \lceil \sqrt{L/2} \rceil$).

Make a [list of baby steps](#):

- ▶ $\mathcal{O}_E, P, 2P, \dots, MP$.

Take [giant steps](#) $Q - MP, Q - 2MP, \dots$ until find a match

$$Q - \lambda_1 MP = \lambda_0 P$$

with the list.

Then $\lambda = \lambda_0 + M\lambda_1$.

- ▶ **Time:** $O(\sqrt{L})$.
- ▶ **Memory:** $O(\sqrt{L})$.

Pollard methods

Use deterministic random walks in $E(\mathbb{F}_q)$:

- ▶ Partition $E(\mathbb{F}_q)$ into 2^n sets G_1, \dots, G_{2^n} .
- ▶ Construct 2^n random points $P_i = \alpha_i P$.
- ▶ Random walk $X \mapsto (X + P_i \text{ if } X \in G_i)$.

Method:

- ▶ Start at $X = P$ and take $O(\sqrt{L})$ steps in random walk and store the final value $Y = \alpha P$.
- ▶ Start at $X = Q$ and take steps in walk until hit Y .
- ▶ Have $Q + \alpha' P = \alpha P$.

- ▶ Time: $O(\sqrt{L})$.
- ▶ Memory: $O(1)$.

Parallel Pollard (Van Oorschot-Wiener)

Distinguished point set \mathcal{D} , size $\theta \# E(\mathbb{F}_q)$.

Method: Suppose we have M processors in parallel.

- ▶ Each processor starts at a random point $X = \alpha P + \beta Q$ and walks in the group.
 - ▶ Every time a distinguished point X is encountered then send (X, α, β) to the central server.
 - ▶ When the server receives (X, α, β) and (X, α', β') then can solve for discrete logarithm.
-
- ▶ Time: $\sqrt{\pi L/2}/M + L/(\theta \# E(\mathbb{F}_q))$.
 - ▶ Server memory: $\theta \sqrt{L}$.

In practice: $L \sim 2^{100}$, $M \sim 2^{10}$ and $\theta = 2^{-30}$.

Equivalence classes (W-Z/G-L-V)

Example :

- ▶ For $P = (x, y)$ have $-P = (x, -y)$.
- ▶ Impose a canonical choice of representative for elements of the set $E(\mathbb{F}_q)/\langle \pm 1 \rangle$
- ▶ Define the random walk on this set instead.
- ▶ Pollard methods are faster by a factor of $\sqrt{2}$.

Example:

- ▶ Consider a subfield curve E/\mathbb{F}_2 with discrete logarithm problem in $E(\mathbb{F}_{2^l})$ (Koblitz curve)
- ▶ Action of $\pm Frob_2$ gives equivalence classes of size $2l$.
- ▶ So method faster by factor $\sqrt{2l}$.

Koblitz curves are recommended in some standards eg ANSI, NIST etc

Special Algorithms

Special Attacks

There are a number of special attacks.

- ▶ Only apply to certain curves
- ▶ Analogous to weak RSA keys

Unlike weak RSA keys, weak ECC keys are easily detected by any user

- ▶ i.e. can be detected by anyone and not just the person who makes the key.

Construct group homomorphism

$$E(\mathbb{F}_q) \longrightarrow \mathbb{F}_{q^k}^*$$

where k is the smallest integer such that the exponent of $E(\mathbb{F}_q)$ divides $q^k - 1$.

Can solve discrete logarithm problem in $\mathbb{F}_{q^k}^*$ using an index calculus algorithm of subexponential complexity (in q^k).

E supersingular implies $k \leq 6$.

General case $k \sim q$.

- ▶ But are some special cases for ordinary curves.

Menezes-Okamoto-Vanstone/Frey-Rück

There is a pairing, the (modified) Tate pairing, such that for supersingular curves

$$t : \begin{cases} E(\mathbb{F}_q) \times E(\mathbb{F}_q) & \longrightarrow \mathbb{F}_{q^k}^* \\ (P, Q) & \longmapsto f_{n,P}(Q)^{(q^k-1)/n} \end{cases}$$

where

- ▶ $(f_{n,P}) = n(P) - n(\mathcal{O})$
- ▶ $n = \#E(\mathbb{F}_q)$
- ▶ $t(P, Q)$ is bilinear
- ▶ If $P, Q \neq \mathcal{O}$ then $t(P, Q) \neq 1$

Menezes-Okamoto-Vanstone/Frey-Rück

To solve

$$Q = \lambda P$$

Compute

- ▶ $g = t(P, P)$
- ▶ $h = t(Q, P)$

Try to solve, in the finite field,

$$\begin{aligned} h &= t(Q, P) \\ &= t(\lambda P, P), \\ &= t(P, P)^\lambda, \\ &= g^\lambda. \end{aligned}$$

Menezes-Okamoto-Vanstone/Frey-Rück

For general curves the modified Tate pairing is defined as

$$t : \begin{cases} E(\mathbb{F}_q) \times \overline{E}(\mathbb{F}_{q^e}) & \longrightarrow \quad \mathbb{F}_{q^k}^* \\ (P, Q) & \longmapsto \quad f_{n,P}(Q)^{(q^k-1)/n} \end{cases}$$

where

- ▶ $(f_{n,P}) = n(P) - n(\mathcal{O})$
- ▶ $n = \#E(\mathbb{F}_q)$
- ▶ $t(P, Q)$ is bilinear
- ▶ If $P, Q \neq \mathcal{O}$ then $t(P, Q) \neq 1$
- ▶ $e = k/d$ where d is the largest possible twist
- ▶ \overline{E} is a d -th twist of E , which is a curve defined over \mathbb{F}_{q^e} .

Suppose $E(\mathbb{F}_p)$ has exactly p points.

Construct a group homomorphism

$$E(\mathbb{F}_p) \longrightarrow \mathbb{F}_p^+.$$

Methods:

- ▶ Using p -adic logarithm and p -adic lift.
- ▶ Take a function f such that $(f) = p(P) - p(O)$ and consider the holomorphic differential $\omega = \frac{1}{f} df$.

Discrete logarithm problem in \mathbb{F}_p^+ solved using Euclid's algorithm.

Weil Descent

Only (currently) applies to fields of characteristic two.

If curve defined over \mathbb{F}_{q^n} for a small value of n can reduce ECDLP to a HCDLP.

For some values of n , eg $n = 4$ this weakens the curve.

- ▶ Work of Frey, Galbraith, Gaudry, Hess and Smart

Values of n of 4, 5, 6 sometimes chosen for efficiency reasons.

Note, only standard which uses such curves is IPSec (I think)

New Techniques

The most successful of the more modern techniques (post 2005) have been those in the Semaev/Gaudry/Diem family.

- ▶ Use a combination of index calculus, division polynomials and Groebner basis.

Almost all are non-practical but they obtain sub-exponential complexity for infinite families of curves over fields of the form

$$\mathbb{F}_{p^n}$$

where p and n lie in certain regions.

- ▶ Sort of “medium characteristic” fields.

Gaudry's Method

For curves over \mathbb{F}_{q^n}

For fixed n , but with q tending to infinity, Gaudry obtains a complexity of

$$O(q^{2-2/n}).$$

Comparing to Pollard rho of $O(q^{n/2})$ we see that for $n = 4$ this is more efficient.

- ▶ But is totally impractical

Diem's Method

If $a > 2 + \epsilon$ and $(2 + \epsilon)n^2 \leq \log_2(q) \leq an^2$ then obtain

$$\exp(O(1) \cdot (\log(q^n))^{2/3})$$

i.e. $L_{q^n}(1/3, c \cdot \sqrt{a})$ for some constant c .

This generalises Gaudry's result.

We essentially obtain a polynomial algorithm in q as long as $\log_2(q)$ is larger than $2n^2$.

- ▶ Hence if q is subexponential in q^n then we get a subexponential algorithm.

Again the method is totally impractical, even for small values of n and q .

How big?

To compare ECC against other technologies we use the following table provided by NIST

Block Cipher Key Size	Example Block Cipher	ECC Key Size	RSA Key Size
80	SKIPJACK	163	1024
128	AES (small)	283	3072
192	AES (medium)	409	7680
256	AES (large)	571	15360

ECC at 571 bits is usable, RSA at 15360 bits is not.

Although 571 is really huge, conservative managers may want to go for the **highest level of security possible**.

Counting Points

Counting points

To use an elliptic curve in a real system we first need to know

$$\#E(\mathbb{F}_q).$$

For some curves this is easy

- ▶ Koblitz curves

For others we need to be more clever to compute this number

Frobenius endomorphism

$$\pi : (x, y) \mapsto (x^q, y^q).$$

Characteristic polynomial

$$\pi^2 - t\pi + q = 0.$$

Then have $\#E(\mathbb{F}_q) = q + 1 - t$.

Hasse: $|t| \leq 2\sqrt{q}$.

Computing $\#E(\mathbb{F}_q)$ is equivalent to computing t .

Idea:

Compute the value of t modulo small primes (or prime powers) /
Recover t using the Chinese remainder theorem and the bound
 $|t| \leq 2\sqrt{q}$.

Very complicated algorithm.

- ▶ Can be made to be very efficient using ideas of Elkies and Atkin
- ▶ Can compute $\#E(\mathbb{F}_q)$ for a given curve in a matter of seconds for most interesting values of q .

Satoh

Satoh in 1998 invented a new method which is better than Schoof for fields \mathbb{F}_{p^n} of small characteristic p , eg characteristic two,

Lifts the curve to a p -adic extension.

Applies a p -isogeny n times, to obtain an isogeny cycle.

Use this to write down t modulo p^n .

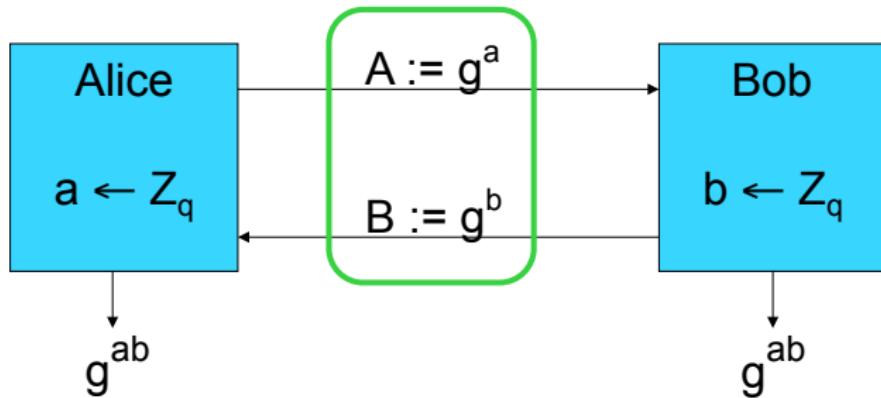
Note : A 2-isogeny is given by the Arithmetic-Geometric mean as any standard textbook on elliptic integrals (or computing π) will tell you.

- ▶ Thus Satoh's algorithm gives rise to the AGM method of Harley-Gaudry from 2001
- ▶ AGM method only useful for characteristic two.

Bilinear Pairing

Recall: Diffie-Hellman protocol

- G: group of prime order q ; $g \in G$ generator



- Security: Decision Diffie-Hellman assumption in G:
 (g, A, B, g^{ab}) indist. from $(g, A, B, g^{\text{rand}})$

Standard complexity assumptions

- G : group of order q ; $1 \neq g \in G$; $x, y, z \leftarrow \mathbb{Z}_q$
 - Discrete-log problem: $g, g^x \Rightarrow x$
-

- Computational Diffie-Hellman problem (CDH):

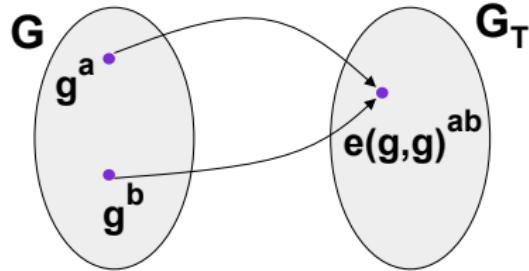
$$g, g^x, g^y \Rightarrow g^{xy}$$

- Decision Diffie-Hellman problem (DDH):

$$g, g^x, g^y, g^z \Rightarrow \begin{cases} 0 & \text{if } z = xy \\ 1 & \text{otherwise} \end{cases}$$

Pairings

- G, G_T : finite cyclic groups of prime order q .



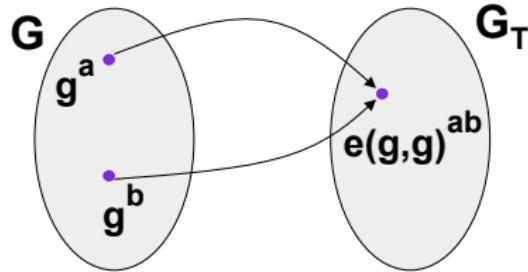
- Def: A **pairing** $e: G \times G \rightarrow G_T$ is a map:

- Bilinear: $e(g^a, g^b) = e(g, g)^{ab} \quad \forall a, b \in \mathbb{Z}, g \in G$
 - Poly-time computable and non-degenerate:
 g generates $G \Rightarrow e(g, g)$ generates G_T

- Current examples: $G \subseteq E(\mathbb{F}_p)$, $G_T \subseteq (\mathbb{F}_{p^\alpha})^*$
 $(\alpha = 1, 2, 3, 4, 6, 10, 12)$

Pairings

- G, G_T : finite cyclic groups of prime order q .



$$e(g^x, h^y) = e(g^y, h^x)$$

- Current examples: $G \subseteq E(\mathbb{F}_p)$, $G_T \subseteq (\mathbb{F}_{p^\alpha})^*$
 $(\alpha = 1, 2, 3, 4, 6, 10, 12)$

Consequences of pairing

- **Decision Diffie-Hellman (DDH)** in G is easy: [J' 00, JN' 01]

- input: $g, g^x, g^y, g^z \in G$

- to test if $z=xy$ do:

$$e(g, g^z) \stackrel{?}{=} e(g^x, g^y)$$

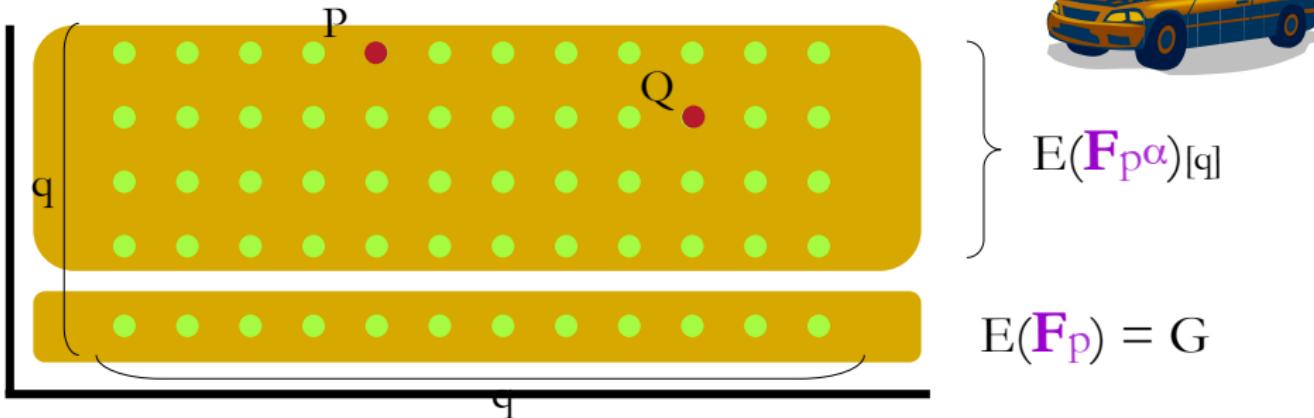
-
- DLog reduction from G to G_T : [MOV '93]

$$\text{DLog}_{\text{in } G} \quad g, g^a \in G \quad \Rightarrow$$
$$\text{DLog}_{\text{in } G_T} \quad e(g,g), e(g,g^a) \in G_T$$

Basic complexity assumptions in bilinear groups

- $e: G \times G \rightarrow G_T$; $1 \neq g \in G$; $x, y, z \leftarrow \mathbb{Z}_q$ ✓
- Discrete-log problem: $g, g^x \Rightarrow x$ ✓
- Computational Diffie-Hellman problem (CDH):
 $g, g^x, g^y \Rightarrow g^{xy}$ ✓
- Bilinear Decision Diffie-Hellman problem (BDDH):
 $h, g, g^x, g^y, e(h, g)^z \Rightarrow \begin{cases} 0 & \text{if } z = xy \\ 1 & \text{otherwise} \end{cases}$

Where pairings come from ...



Tate pairing: $e(P, Q) := f_P(Q)^{(p^{\alpha}-1)/q}$, $(f_P) = q \cdot (P) - q \cdot (O)$

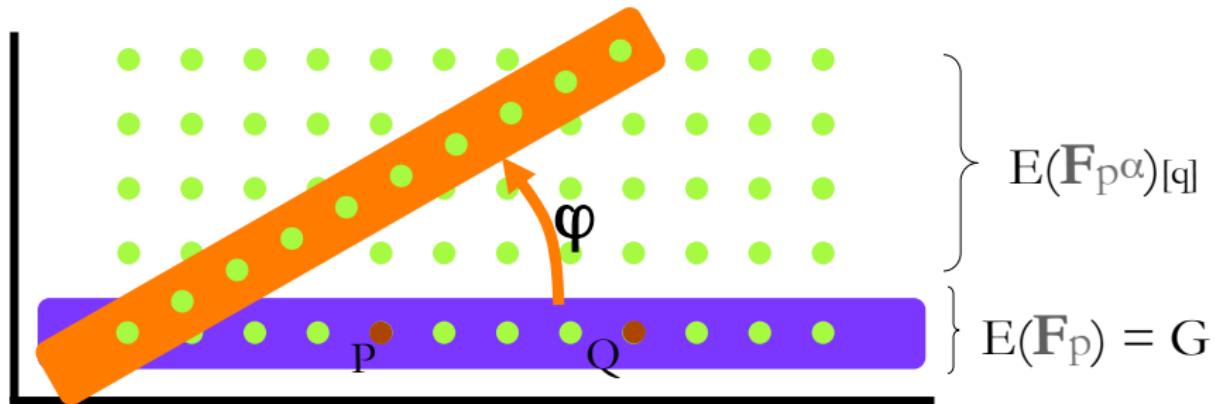
V. Miller (84): f_P has a short straight line program

... but: $\forall P, Q \in G : e(P, Q) = 1$

Supersingular bilinear groups

Supersingular curves:

$$(\text{e.g. } y^2 = x^3 + x , \quad p=3 \pmod{4})$$



$$\overline{\mathbf{e}} : G \times G \rightarrow G_T$$

$$\text{Def: } \overline{\mathbf{e}}(P, Q) = \mathbf{e}(P, \varphi(Q))$$

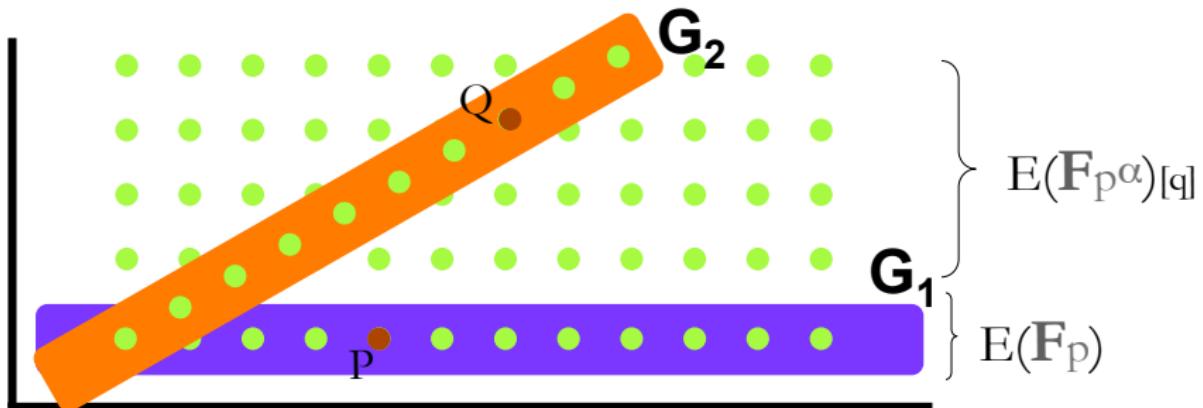
Possible α : $\alpha=2,3,4,6$ or “ α =7.5 [RS '02]

Asymmetric pairings

$$e: G_1 \times G_2 \rightarrow G_T$$

Non-supersingular curves: (1st case)

$$(G_1 \neq G_2)$$



No mapping φ out of $E(F_p)$

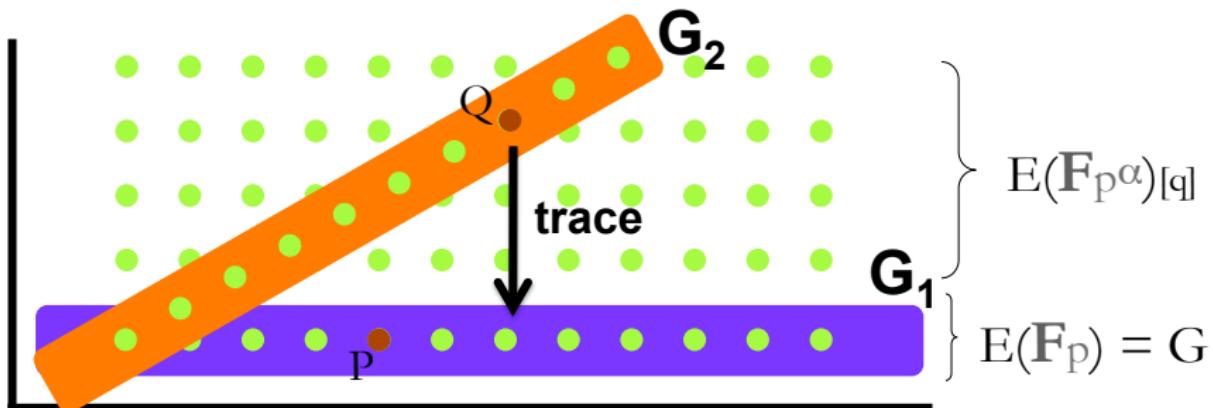
$$e: G_1 \times G_2 \rightarrow G_T$$

Asymmetric pairings

$$e: G_1 \times G_2 \rightarrow G_T$$

Non-supersingular curves: (1st case)

$$(G_1 \neq G_2)$$



Projection map $\text{tr}: G_2 \rightarrow G_1 \quad \Rightarrow$

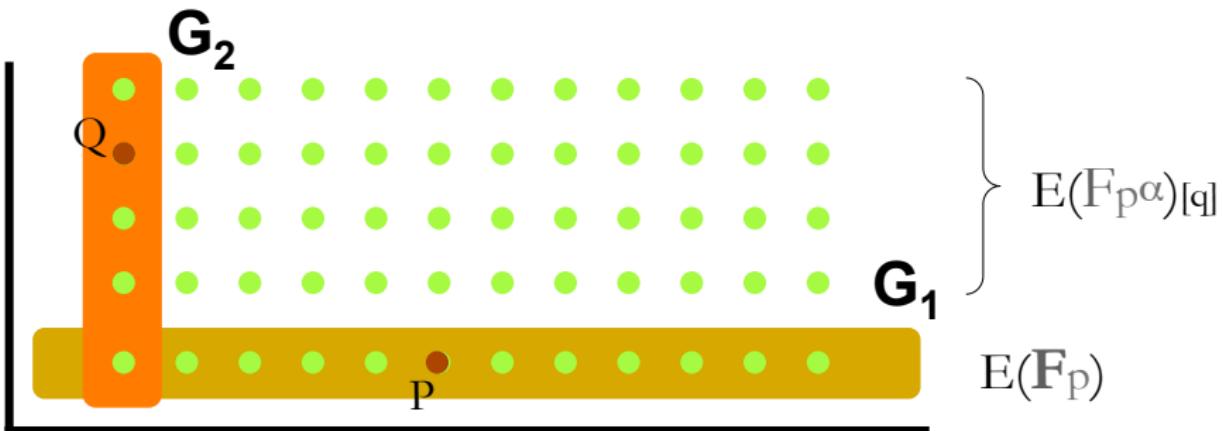
Symmetric pairing on $G_2 \Rightarrow$ easy DDH in G_2

... but no (known) DDH algorithm in G_1

Asymmetric pairings

$$e: \mathbf{G}_1 \times \mathbf{G}_2 \rightarrow \mathbf{G}_T$$

Non-supersingular curves: (2nd case)



No projection map \Rightarrow no known DDH algorithm in \mathbf{G}_1 or \mathbf{G}_2

SXDH assumption: DDH hard in \mathbf{G}_1 and \mathbf{G}_2

- Used for anonymous IBE, circular insecure enc., ...

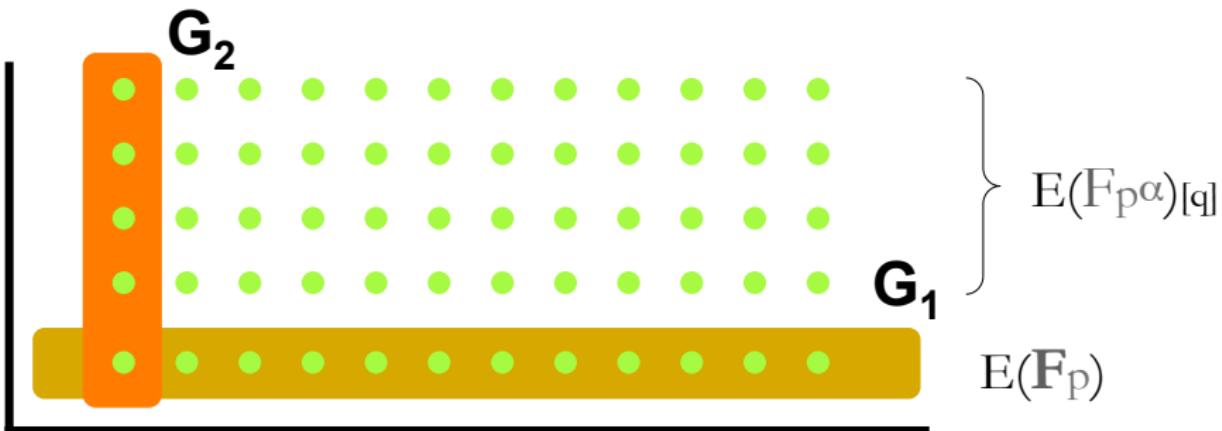
[D'10]

[ABBC'10, CGH'12]

Asymmetric pairings

$$e: \mathbf{G}_1 \times \mathbf{G}_2 \rightarrow \mathbf{G}_T$$

Non-supersingular curves: (2nd case)



Most efficient implementations

MNT and BN groups: asymmetric pairings

G_2

Open problem: larger α (prime order $E(\mathbf{F}_p)$)

e.g. $\alpha = 16, 20, 24, \dots$ (see taxonomy [FST'10])

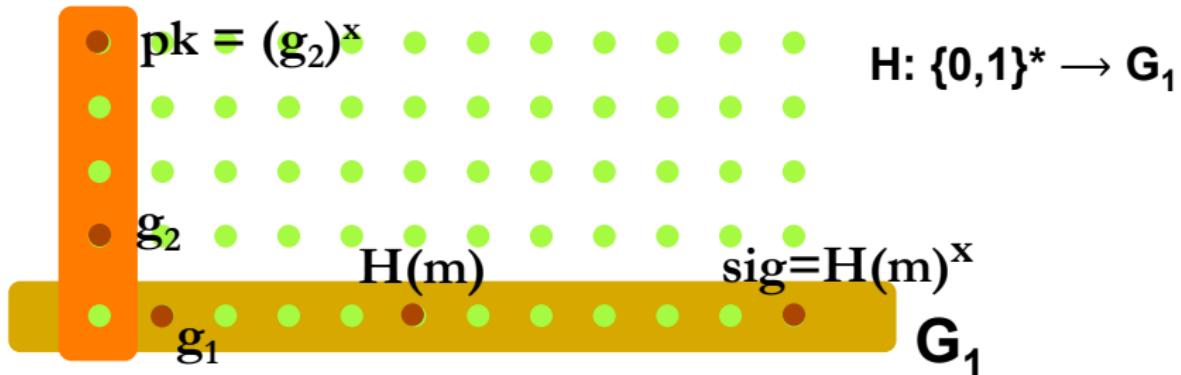


$$E(\mathbf{F}_p) = G_1$$

$$\mathbf{e} : G_1 \times G_2 \rightarrow G_T$$

- MNT '01 Curves: $\alpha=2,3,4,6$
 - BN '05, F'05 Curves: $\alpha=10, 12$
- } not supersingular curves

Example: BLS sigs. using asymmetric pairings



KeyGen: output $[g_1, g_2, pk=(g_2)^x]$, $sk \leftarrow x$

Sign(sk, m): output $\text{sig} \leftarrow H(m)^x \in G_1$

Verify(pk, m, s): accept iff $e(H(m), pk) \stackrel{?}{=} e(\text{sig}, g_2)$

Security: EUF-CMA assuming aCDH (in RO model)

$$g_2, g_2^x, g_1, g_1^x, g_1^y \not\Rightarrow g_1^{xy}$$

More complexity assumptions in bilinear groups

The decision linear assumption (DLIN) [BBS'04]

The **k-DLIN** assumption in G: (prime order q)

$$\left[\begin{array}{c} g_1, g_2, \dots, g_k, g_{k+1} \\ g_1^{x_1}, g_2^{x_2}, \dots, g_k^{x_k}, (g_{k+1})^{\sum x_i} \end{array} \right] \approx_p \left[\begin{array}{c} g_1, g_2, \dots, g_k, g_{k+1} \\ g_1^{x_1}, g_2^{x_2}, \dots, g_k^{x_k}, (g_{k+1})^y \end{array} \right]$$

Hierarchy: DDH \equiv **1-DLIN** \geq **2-DLIN** $\geq \dots \geq$ **k-DLIN** $\geq \dots$

“easiest” to break

“harder”
to break

Fact: $(k+1)$ -linear map in G \Rightarrow k-DLIN is false (homework)

Assumption: k-DLIN holds even if k' -linear map in G for $k' \leq k$

The decision linear assumption (DLIN)

- Many bilinear constructions can be based on 2-DLIN
- A useful implication: $g \in G$ order q

k-DLIN \Rightarrow $(k < n, m)$

$$A \xleftarrow{R} (Z_q)^{n \times m}$$

output g^A

 \approx_p

$$B \xleftarrow{R} (Z_q)^{n \times m}, \text{ rank}(B)=k$$

output g^B

The “master” assumption [BBG’04]

Let $\{f\}$, $F = \{f_0=1, f_1, f_2, \dots, f_n\} \subseteq F_q[x_1, \dots, x_m]$

such that $f \notin \text{span}_{F_q}(\{f_i \cdot f_j / f_k\}_{i,j,k})$ (*)

The (F, f) assumption: in a bilinear group G of order q

$$g^{f_1(\bar{x})}, \dots, g^{f_n(\bar{x})}, g^{\mathbf{f}(\bar{x})}$$

\approx_p

$$g^{f_1(\bar{x})}, \dots, g^{f_n(\bar{x})}, g^{\mathbf{y}}$$

Thm (informal): $\forall (F, f)$ satisfying (*) and poly. degree,
the (F, f) assumption holds in a **generic** bilinear group

Composite order groups

Bilinear groups of order $N=pq$

[BGN' 05]

- G : group of order $N=pq$. **(p, q) – secret**
bilinear map: $e: G \times G \rightarrow G_T$

$$G = G_p \times G_q . \quad g_p = g^q \in G_p \quad ; \quad g_q = g^p \in G_q$$

- Facts: $e(g_p, g_q) = e(g^q, g^p) = e(g, g)^N = 1$

$$e(g_p, g_p^x \cdot g_q^y) = e(g_p, g_p)^x$$

An example: BGN encryption [BGN'05]

- KeyGen(λ): generate bilinear group G of order $N=p \cdot q$

$$pk \leftarrow (G, N, g, g_p) ; sk \leftarrow p$$

- Enc(pk, m) : $r \leftarrow Z_N , C \leftarrow g^m (g_p)^r \in G$

- Dec(sk, C) : $C^p = [g^m]^p \cdot [g_p^r]^p = (g_q)^m \in G_q$

Output: $Dlog_{g_q}(C^p)$

- Note: decryption time is $O(\sqrt{m})$
⇒ require small message space (e.g. $\{0,1\}$)

Homomorphic Properties

$$C_1 \leftarrow g^{m_1} (g_p)^{r_1}, \quad C_2 \leftarrow g^{m_2} (g_p)^{r_2} \in G$$

- Additive hom: $E(m_1 + m_2) = C_1 \cdot C_2 \cdot (g_p)^s$
- One mult hom: $\hat{E}(m_1 \cdot m_2) = e(C_1, C_2) \cdot e(g_p, g_p)^s$

More generally: $E(m_1), \dots, E(m_n) \rightarrow \hat{E}(F(m_1, \dots, m_n))$

For any $F \in Z_N[X_1, \dots, X_n]$ of total degree 2

Example: matrix-matrix product of encrypted matrices [AW'07]
(becomes fully homomorphic with a k-linear map, for suff. large k)

Security: the subgroup assumption

Subgroup assumption:

$$\mathbf{G} \approx \mathbf{G}_p$$

Distribution $\mathbf{P}_G(\lambda)$:

$(G, g, p, q) \leftarrow \text{GroupGen}(\lambda)$

$N \leftarrow p \cdot q$

$s \leftarrow Z_N$

Output: (G, g, N, \mathbf{g}^s)

Distribution $\mathbf{P}_p(\lambda)$:

$(G, g, p, q) \leftarrow \text{GroupGen}(\lambda)$

$N \leftarrow p \cdot q$

$s \leftarrow Z_N$

Output: $(G, g, N, (\mathbf{g}_p)^s)$

For any poly-time A:

$$|\Pr[A(X) : X \leftarrow \mathbf{P}_G(\lambda)] - \Pr[A(X) : X \leftarrow \mathbf{P}_p(\lambda)]| < \text{neg}(\lambda)$$

Thm: BGN is semantically secure under the subgroup assumption

From composite order to prime order

A general conversion: [F'10, L'12]

composite order bilinear groups system

⇒ prime order bilinear group based on 2-DLIN

- Resulting systems are often more efficient
(since group size is smaller)
but are technically more complex

Final note: pairings mod N

Consider elliptic curve $E: y^2 = x^3 + ax + b \pmod{N}$

where $N=p \cdot q$ is an RSA modulus

Then $E(\mathbb{Z}/N\mathbb{Z}) = E(\mathbb{F}_p) \times E(\mathbb{F}_q)$

- Finding size of $E(\mathbb{Z}/N\mathbb{Z})$ is as hard as factoring N
 - ⇒ cannot compute pairings on E
 - ⇒ no known algorithm for DDH on $E(\mathbb{Z}/N\mathbb{Z})$
- But DDH becomes easy given p, q
 - ⇒ trapdoor DDH group

Identity-based Encryption

Recall: Pub-Key Encryption (PKE)

PKE Three algorithms : (G, E, D)

$G(\lambda) \rightarrow (\text{pk}, \text{sk})$ outputs pub-key and secret-key

$E(\text{pk}, m) \rightarrow c$ encrypt m using pub-key pk

$D(\text{sk}, c) \rightarrow m$ decrypt c using sk

obtain
 pk_{alice}



$E(\text{pk}_{\text{alice}}, \text{msg})$



Example: ElGamal encryption

- $G(\lambda): (G, g, q) \leftarrow \text{GenGroup}(\lambda)$

$$\text{sk} := (\alpha \leftarrow F_p) \quad ; \quad \text{pk} := (h \leftarrow g^\alpha)$$

- $E(\text{pk}, m \in G): s \leftarrow Z_q \text{ and do } c \leftarrow (g^s, m \cdot h^s)$

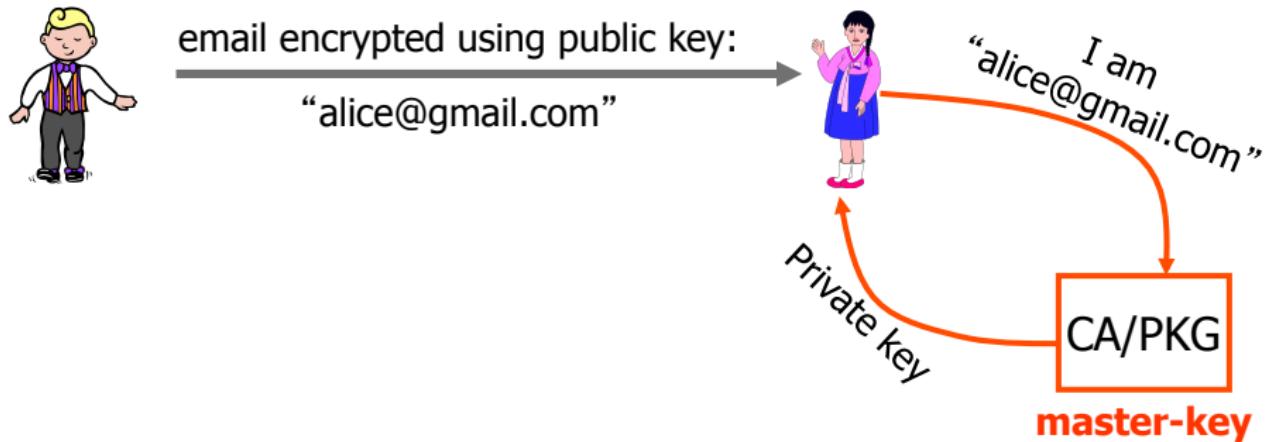
- $D(\text{sk}=\alpha, c=(c_1, c_2)): \text{observe } c_1^\alpha = (g^s)^\alpha = h^s$

- Security (IND-CPA) based on the DDH assumption:

(g, h, g^s, h^s) indist. from $(g, h, g^s, g^{\text{rand}})$

Identity Based Encryption [Sha '84]

- IBE: PKE system where PK is an arbitrary string
 - e.g. e-mail address, phone number, IP addr...



Identity Based Encryption

[Sha '84]

Four algorithms : (S,K,E,D)

$S(\lambda) \rightarrow (\text{pp}, \text{mk})$ output params, pp,
and master-key, mk

$K(\text{mk}, \text{ID}) \rightarrow d_{\text{ID}}$ outputs private key, d_{ID} , for ID

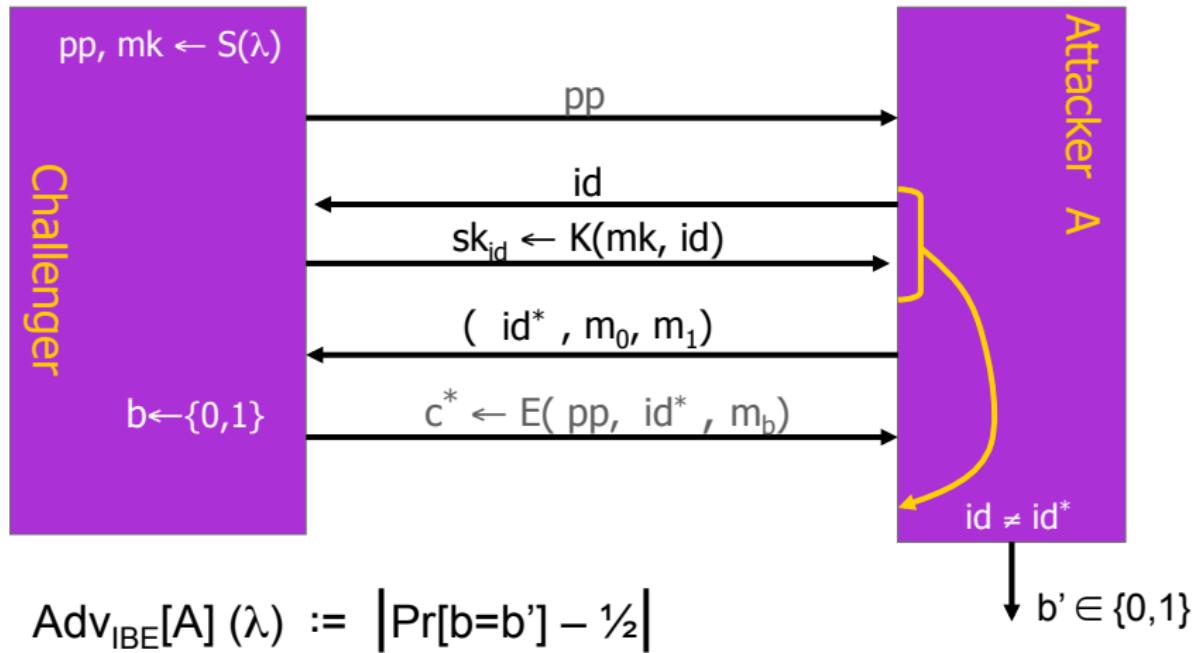
$E(\text{pp}, \text{ID}, m) \rightarrow c$ encrypt m using pub-key ID (and pp)

$D(d_{\text{ID}}, c) \rightarrow m$ decrypt c using d_{ID}

IBE “compresses” exponentially many pk's into a short pp

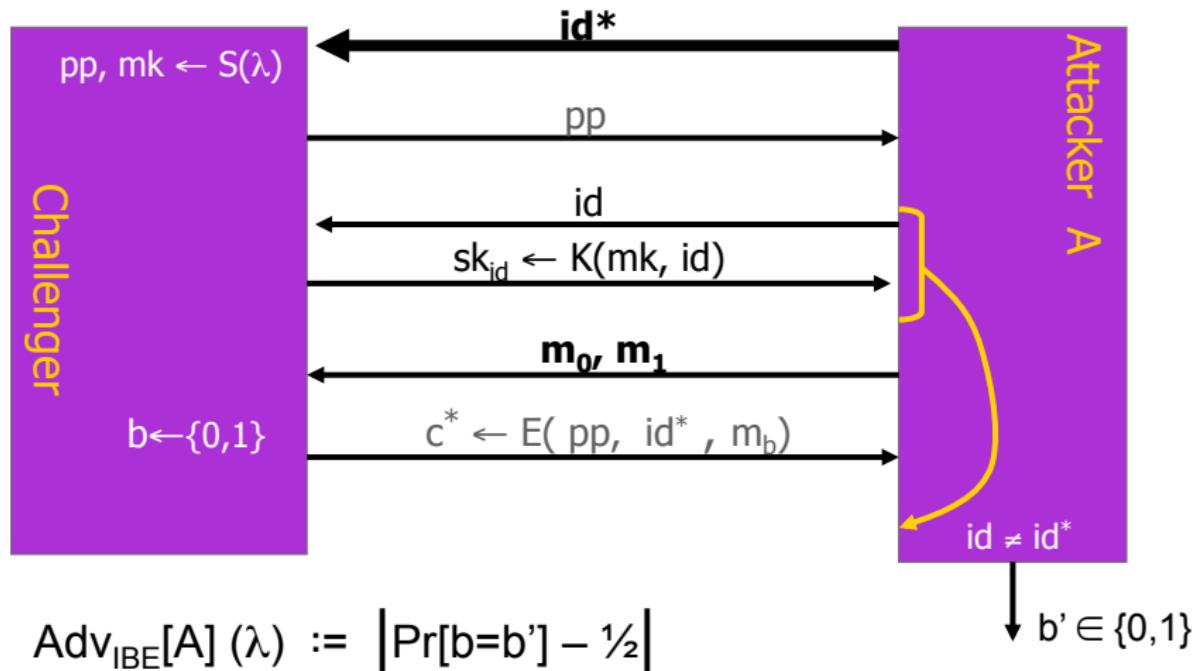
CPA-Secure IBE systems (IND-IDCPA) [B-Franklin'01]

Semantic security when attacker has few private keys

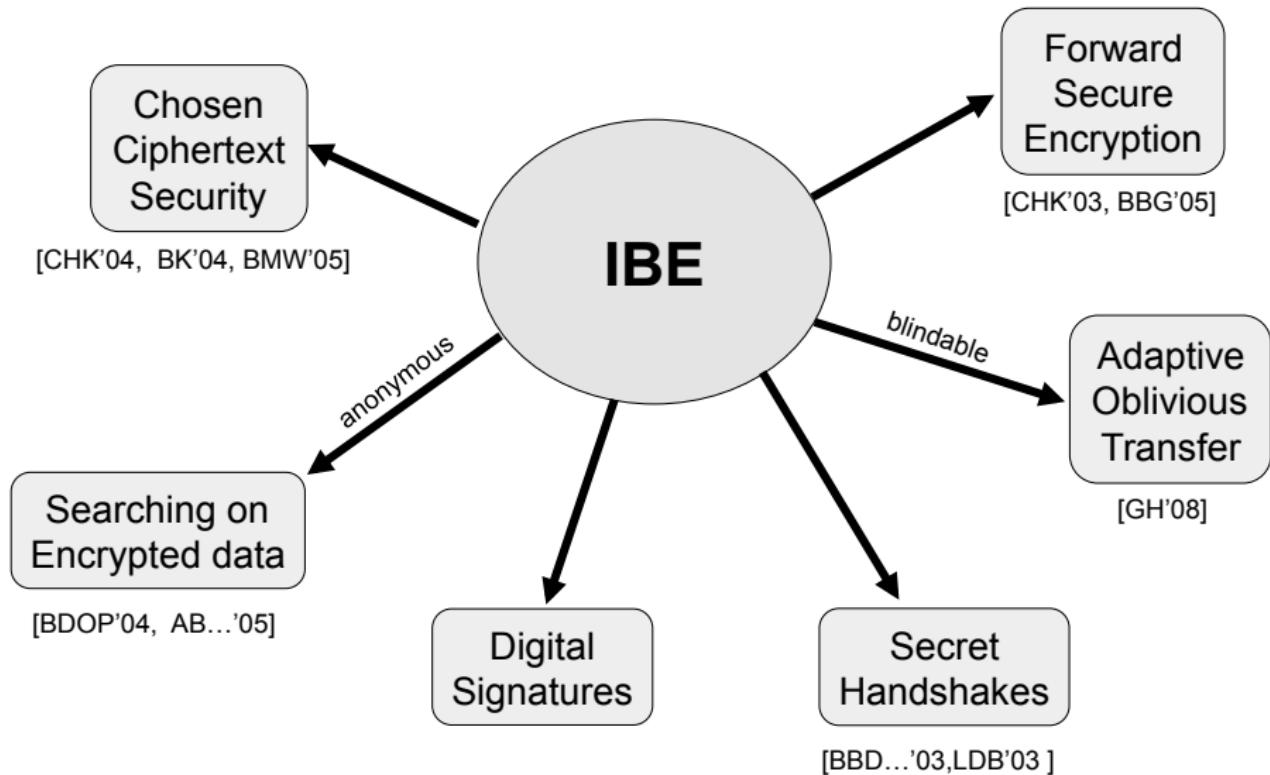


CPA-Secure IBE systems (IND-sIDCPA) [CHK'04]

Selective security: commit to target \mathbf{id}^* in advance



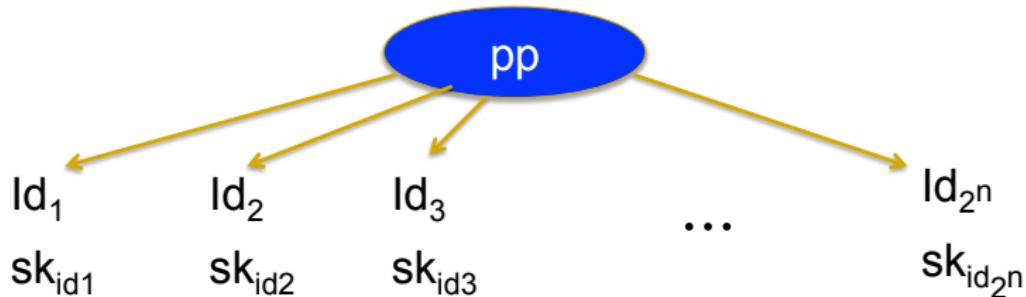
Why ID Based Encryption?



Black box separation [BPRVW'08]



Main reason: short pp defines exp. many public keys

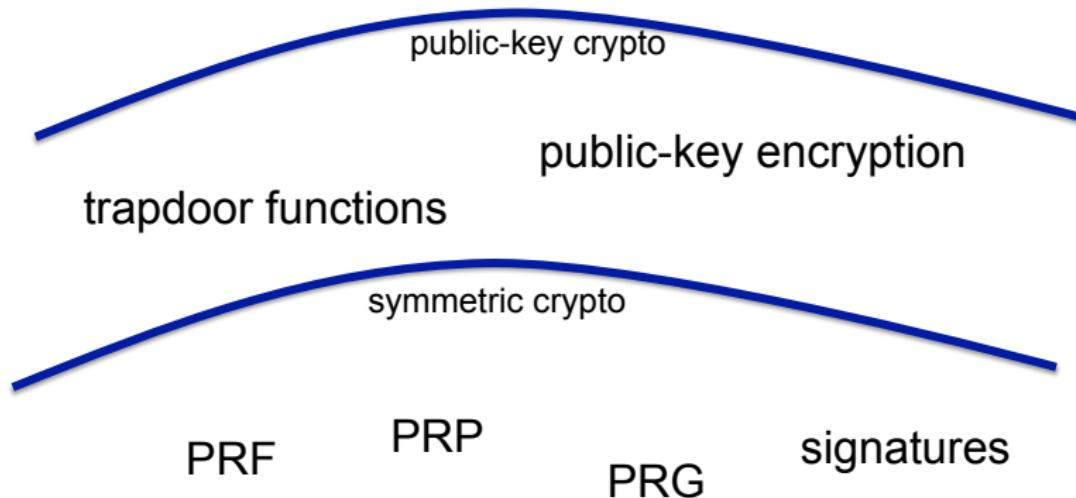


Functional encryption [BSW'11]

ABE [SW'05]

Hierarchical IBE [HL'02, GS'02]

IBE



IBE in practice

Bob encrypts message with pub-key:

“alice@hotmail || role=accounting || time=week-num”

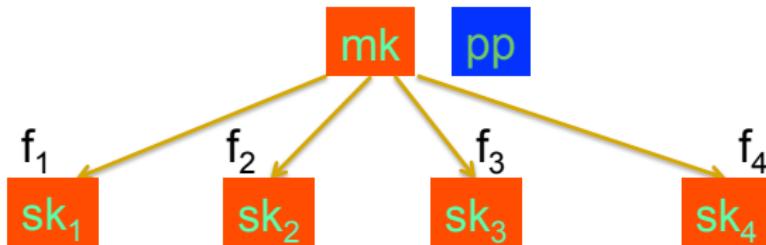
policy-based encryption

short-lived keys



Aug. 2011: “... Voltage SecureMail ... with over one billion secure business emails sent annually and over 50 million worldwide users.”

IBE: functional encryption view [BSW'11]



$$E(pp, \text{data}) , \text{ sk}_1 \Rightarrow f_1(\text{data})$$

IBE: first non-trivial functionality

$$E(pp, \underbrace{(\text{id}_0, m)}_{\text{data}}) , \text{ sk}_{\text{id}} \Rightarrow \text{output} \begin{cases} m & \text{if } \text{id} = \text{id}_0 \\ \perp & \text{otherwise} \end{cases}$$

Constructing IBE

Can we build an IBE ??

- ElGamal is not an IBE:

$$\text{sk} := (\alpha \leftarrow F_p) \quad ; \quad \text{pk} := (h \leftarrow g^\alpha)$$

- pk can be any string: $\textcolor{blue}{h} = \underline{\textit{alice@gmail.com}}$ $\in \mathbf{G}$

... but cannot compute secret key α

- Attempts using trapdoor Dlog [MY'92] but inefficient

Can we build an IBE ??

- RSA is not an IBE:

$$\text{pk} := (\text{N} = p \cdot q, e) \quad ; \quad \text{sk} := (d)$$

- Cannot map ID to (N,e)
- How about: fix N and and use $e_{\text{id}} = \text{Hash}(\text{id})$
- Problem: given $(N, e_{\text{id}}, d_{\text{id}})$ can factor N

IBE Constructions: three families

	Pairings $e: G \times G \rightarrow G'$	Lattices (LWE)	Quadratic Residuosity
IBE w/RO	BF'01	GPV'08	Cocks'01 BGH'07
IBE no RO	CHK'03, BB'04, W'05, G'06, W'09, ... ↔ ↔ ↔	CHKP'10, ABB'10 , MP'12 ??	??
HIBE	GS'03, BB'04 BBG'05, GH'09, LW'10, ...	CHKP'10, ABB'10 ABB'10a	??
extensions	many	some	??

Pairing-based constructions

Some pairing-based IBE constructions

- **BF-IBE** [BF'01]: $\text{BDH} \Rightarrow \text{IND-IDCPA}$ (in RO model)
- **BB-IBE** [BB'04]: $\text{BDDH} \Rightarrow \text{IND-sIDCPA}$
- **Waters-IBE** [W'05]: generalizes BB-IBE
 $\text{BDDH} \Rightarrow \text{IND-IDCPA}$ (but long PP)
- **Gentry-IBE** [G'06]: $\text{q-BDHE} \Rightarrow \text{IND-IDCPA}$ and short PP
- **DualSys-IBE** [W'09]: $\text{2-DLIN} \Rightarrow \text{IND-IDCPA}$ and short PP

BF-IBE: IBE in the RO model

[BF'01]

- $S(\lambda)$: $(G, G_T, g, q) \leftarrow \text{GenBilGroup}(\lambda)$, $\alpha \leftarrow F_p$

$$pp := [g, \ y \leftarrow g^\alpha] \in G \quad ; \quad mk := \alpha$$

- $K(mk, id)$: $sk \leftarrow H(id)^\alpha$

- $E(pp, id, m)$: $s \leftarrow F_p$ and do

$$C \leftarrow (g^s, m \cdot e(y, H(id))^s) \stackrel{\approx}{\equiv} e(g^s, H(id)^s)$$

- #### ■ D(sk, (c₁, c₂)):

observe: $e(c_1, d) = e(g^s, H(id)^\alpha)$

IBE w/o RO: a generic approach

The “all but one” paradigm

real system

Generate “**real**” PP
and “**real**” MK

MK: generate SK_{ID} for all id



simulation

Given id^* do:
Generate “**fake**” PP’
and “**fake**” MK’

MK’: all but one ($\text{id} \neq \text{id}^*$)



challenge CT for id^*
solves some hard problem

BB-IBE: IBE w/o random oracles [BB'04]

- $S(\lambda)$: $(G, G_T, g, q) \leftarrow \text{GenBilGroup}(\lambda), \quad \alpha \leftarrow F_p$
 $pp := [g, \ y \leftarrow g^\alpha, \ g_1, \ h] \in G \quad ; \quad mk := g_1^\alpha$
- $K(mk, id)$: $sk_{ID} \leftarrow (mk \cdot (y^{id} \cdot h)^r, \ g^r)$
 $r \leftarrow F_p$
- $E(pp, id, m)$: $s \leftarrow F_p$ and do
 $C \leftarrow (g^s, \ (y^{id} \cdot h)^s, \ m \cdot e(y, g_1)^s)$
- $D(d_1, d_2, c_1, c_2, c_3)$:
observe: $e(c_1, d_1) / e(c_2, d_2) = e(y, g_1)^s$

Security: the all-but-one paradigm

real system

$$\alpha \leftarrow F_p$$

$$g_1, h \leftarrow G$$

$$\text{PP: } g, y \leftarrow g^\alpha, g_1, h \in G$$

$$\text{MK: } g_1^\alpha \in G$$

$$\text{sk}_{\text{id}} \leftarrow (\text{mk} \cdot (y^{\text{id}} \cdot h)^r, g^r)$$



$\text{id} = \text{id}^*$

simulation (all but one)

Given id^* do:

$$\beta \leftarrow F_p$$

$$g_1, y \leftarrow G$$

$$\text{PP: } g, y, g_1, h \leftarrow y^{-\text{id}^*} \cdot g^\beta$$

$$\text{MK': } \beta \in F_p$$

$$\text{sk}_{\text{id}} \leftarrow (d_0, d_1)$$

$$\begin{cases} d_0 = g_1^{-\beta / (\text{id} - \text{id}^*)} \cdot (y^{\text{id}} \cdot h)^r \\ d_1 = g_1^{-1 / (\text{id} - \text{id}^*)} \cdot g^r \end{cases}$$

Reduction to BDDH

$$(g, y, g^r, g^s, e(y,g)^{rs}) \approx_p (g, y, g^r, g^s, R)$$

Given challenge $(g, y, g_1=g^r, z=g^s, T)$ do:

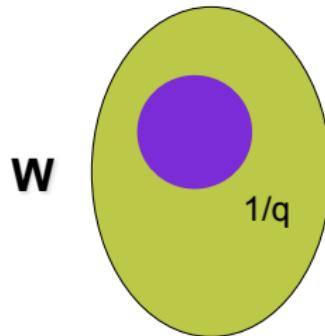
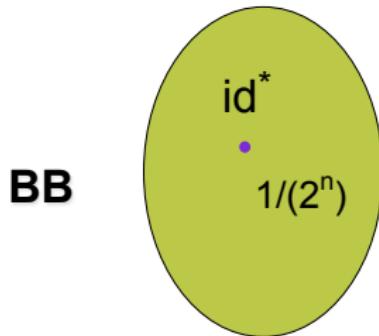
$$pp = (g, y, g_1, h \leftarrow y^{-id^*} \cdot g^\beta) ; \quad mk' = \beta \in F_p$$

- Respond to adversary queries for $id \neq id^*$ using β
- Challenge ciphertext for id^* is

$$\begin{aligned} ct^* &\leftarrow (g^s, (y^{id} \cdot h)^s, m \cdot e(y, g_1)^s) = \\ &= (g^s, (g^\beta)^s, m \cdot e(y, g)^{rs}) = \\ &= (z, z^\beta, m \cdot R) \end{aligned}$$

From selective to full security

[Wat '05]



$$\text{id} = \text{id}^*$$

“all but one”

$$a_1\text{id}_1 + \dots + a_n\text{id}_n = v$$

“all but many”

($q = \#$ private key queries from adv.)

With prob. $\approx(1/q)$:

all queries are “green”, but challenge id^* is blue

The system [Wat '05]

- G(λ): $\alpha \leftarrow F_p$, $g_1, h, y_1, \dots, y_n \leftarrow G$
 $pp = (g, g_1, y \leftarrow g^\alpha, h, y_1, \dots, y_n) \in G$, $mk = g_1^\alpha$

- K(sk, id, m): $r \leftarrow F_p$, $id = b_1 b_2 \dots b_n \in \{0,1\}^n$

$$sk_{id} \leftarrow (mk \cdot (y_1^{b_1} y_2^{b_2} \dots y_n^{b_n} \cdot h)^r, g^r) \in G^2$$

- E(pk, id=b₁b₂...b_n, m):
 $e(c_1, g) / e(y_1^{b_1} \dots y_n^{b_n} \cdot h, c_2) = e(g_1, y)$

Gentry's IBE

[Gen'06]

- S(λ): $(G, G_T, g, q) \leftarrow \text{GenBilGroup}(\lambda)$, $\alpha \leftarrow F_p$

$$pp := [g, y \leftarrow g^\alpha, h] \in G \quad ; \quad mk := \alpha$$

- K(mk, id): $sk \leftarrow [r \leftarrow F_p, (h g^{-r})^{1/(\alpha-\text{id})}]$

- E(pp, id, m): $s \leftarrow F_p$ and do

$$C \leftarrow (y^s \cdot g^{-s \cdot \text{id}}, e(g, g)^s, m \cdot e(g, h)^{-s})$$

\Downarrow
 $g^{s(\alpha-\text{id})}$

Proof idea (IND-IDCPA security)

Simulator knows one private key for every ID

⇒ can respond to all private key queries

⇒ tight reduction to hardness assumption

Hardness assumption (simplified): **q-BDDH** [MSK'02, BB'04, ...]

given $g, v, g^\alpha, g^{(\alpha^2)}, \dots, g^{(\alpha^q)}$:

$$e(g, v)^{(\alpha^{q+1})} \approx_p \text{uniform}(G_T)$$

($q = \#$ private key queries from adv.)

note: challenge ct always decrypts correctly under simulator's secret key

Proof idea (IND-IDCPA security)

Simulator: given $g, v, g^\alpha, g^{(\alpha^2)}, \dots, g^{(\alpha^q)}$

■ **Setup:** rand. poly. $f(x) = a_0 + a_1 \cdot x + \dots + a_q x^q \in F_p[x]$

give adv. $PP := (g, y=g^\alpha, h \leftarrow g^{f(\alpha)})$

■ **Queries:** need $sk = [r \leftarrow F_p, (h g^{-r})^{1/(\alpha-\text{id})}]$

do: $r \leftarrow f(\text{id})$, $(h g^{-r})^{1/(\alpha-\text{id})} = g^{[f(\alpha) - f(\text{id})] / (\alpha - \text{id})}$

and observe that $[f(x) - f(\text{id})] / (x - \text{id}) \in F_p[x]$

DualSys IBE [Wat'09, LW'10]

Covered in Allison Lewko's lecture

New Signature Systems

CDH \Rightarrow short and efficient sigs (!!)

IBE \Rightarrow Simple digital Signatures

[N' 01]

- $\text{Sign}(\text{MK}, m)$: $\text{sig} \leftarrow K(\text{MK}, m)$
- $\text{Verify}(\text{PP}, m, \text{sig})$: Test that **sig** decrypts messages encrypted using m

- Conversely: which sig. systems give an IBE?

- Rabin signatures: [Cocks' 01, BGH' 07]
- GPV signatures: [GPV'09]
- Open problem: IBE from GMR, GHR, CS, ...

Signatures w/o Random Oracles

Example: signature system from BB-IBE (selectively secure)

- $G(\lambda)$: $\alpha \leftarrow F_p, g_1, h \leftarrow G$
 $pk = (g, g_1, y \leftarrow g^\alpha, h) \in G^4, sk = g_1^\alpha$

- Sign(sk, m): $r \leftarrow F_p,$
 $s \leftarrow (sk \cdot (y^m h)^r, g^r) \in G^2$

- Verify($pk, m, S=(s_1, s_2)$): $e(s_1, g) / e(y^m h, s_2) \stackrel{?}{=} e(g_1, y)$

Waters Sigs: existentially unforgeable [Wat '05]

- G(λ): $\alpha \leftarrow F_p$, $g_1, h, y_1, \dots, y_n \leftarrow G$
 $pk = (g, g_1, y \leftarrow g^\alpha, h, y_1, \dots, y_n) \in G$, $sk = g_1^\alpha$

- Sign(sk, m): $r \leftarrow F_p$, $m = m_1 m_2 \dots m_n \in \{0,1\}^n$

$$S \leftarrow (sk \cdot (y_1^{m_1} y_2^{m_2} \dots y_n^{m_n} \cdot h)^r, g^r) \in G^2$$

- Verify($pk, m, S = (s_1, s_2)$):

$$e(s_1, g) / e(y_1^{m_1} \dots y_n^{m_n}, h, s_2) = e(g_1, y)$$

Summary thus far

IBE from pairings:

- BDDH or 2-DLIN \Rightarrow efficient secure IBE

Short signatures from pairings:

- CDH \Rightarrow existential unforgeability
- with RO: $\text{sig} \in \mathbf{G}$, without RO: $\text{sig} \in \mathbf{G}^2$
[BLS'01] [Wat'05]

Anonymous IBE

Simplest non-trivial example of
“private index functional encryption”

Anonymous IBE [BDOP'04, AB...'05, BW'05, ...]

Goal: IBE ciphertext $E(pp, id, m)$

should reveal no info about recipient id

Why?

- A natural security goal
- More importantly, enables searching on enc. Data

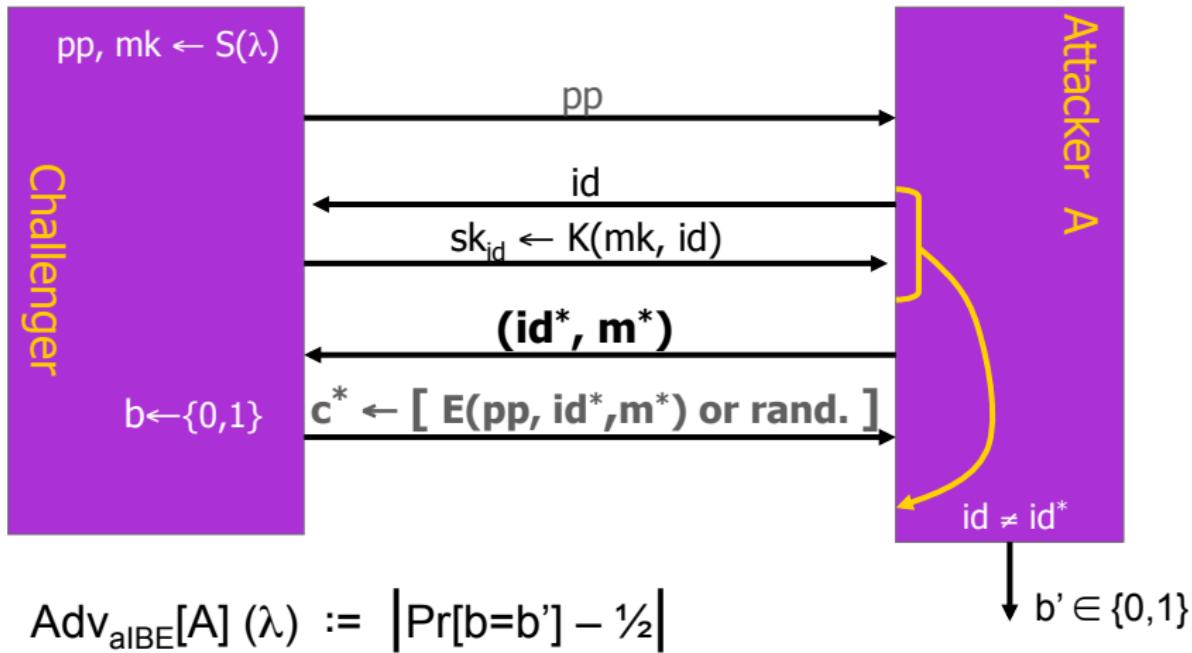
Constructions:

- RO model: BF-IBE
- std. model: 2-DLIN [BW'06],
composite order groups [BW'07] , and SXDH [D'10]

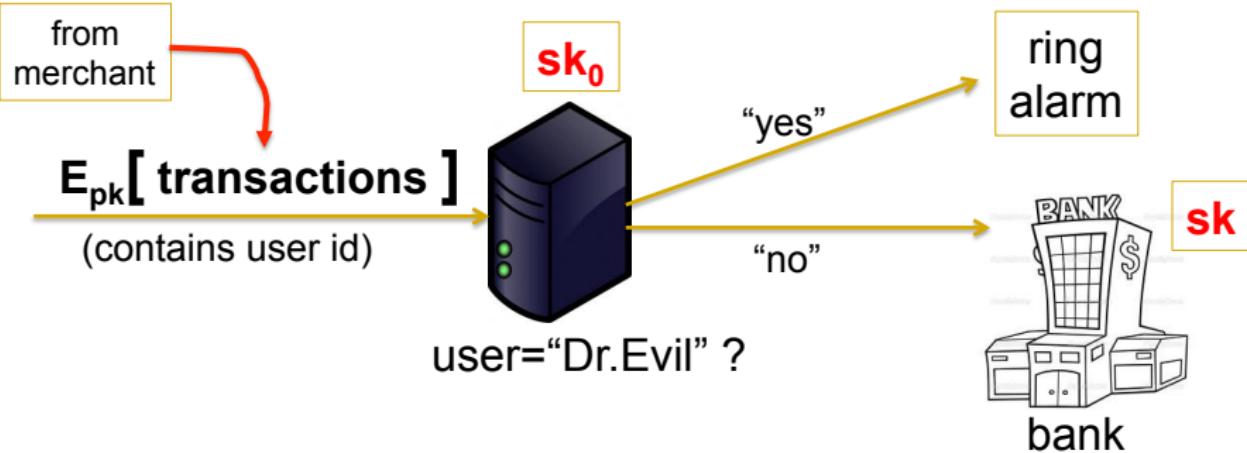
also many lattice-based constructions [GPV'09, CHKP'10, ABB'10]

Anon. IBE systems (anonIND-IDCPA)

Semantic security when attacker has few private keys



Anon. IBE \Rightarrow Basic searching on enc. data



Proxy needs key that lets it test “user=Dr.Evil” and nothing else.

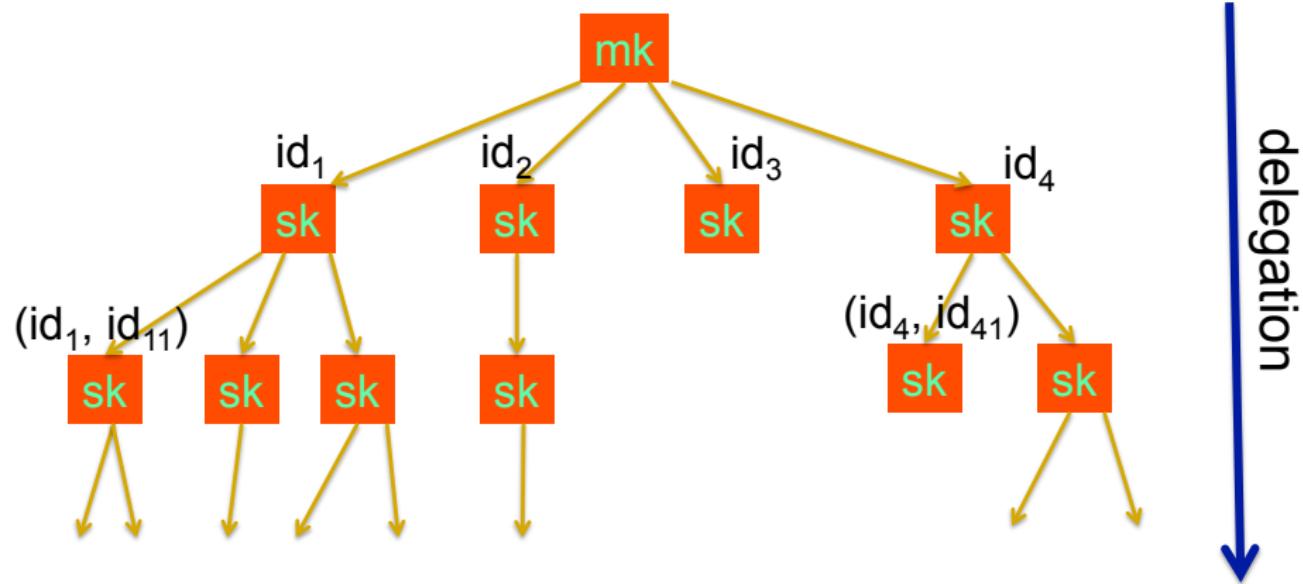
Merchant: embed $c \leftarrow E(pp, user, 1)$ in ciphertext

Proxy: has $sk_0 \leftarrow K(sk, "Dr.Evil")$; tests $D(sk_0, c) \stackrel{?}{=} 1$

Hierarchical IBE

Hierarchical IBE

[HL'02, GS'02, BBG'04, ...]



- Can encrypt a message to $\text{id} = (\text{id}_1, \text{id}_{11}, \text{id}_{111})$
- Only sk_{id} and parents can decrypt
 - Coalition of other nodes learns nothing

Some pairing-based HIBEs

- **GS-HIBE** [GS'03]: $\text{BDH} \Rightarrow \text{IND-IDCPA}$ (in RO model)
 - **BB-HIBE** [BB'04]: $\text{BDDH} \Rightarrow \text{IND-sIDCPA}$
 - **BW-HIBE** [BW'05]: $\text{2-DLIN} \Rightarrow \text{anonIND-sIDCPA}$
- ⇒ ciphertext size grows linearly with hierarchy depth
- ⇒ adaptive security: sec. degrades exp. in hierarchy depth

Some pairing-based HIBEs

- **GS-HIBE** [GS'03]: BDH \Rightarrow IND-IDCPA (in RO model)
 - **BB-HIBE** [BB'04]: BDDH \Rightarrow IND-**sIDCPA**
 - **BW-HIBE** [BW'05]: 2-DLIN \Rightarrow anonIND-**sIDCPA**
 - **BBG-HIBE** [BBG'05]: d-BDDH \Rightarrow IND-**sIDCPA**
 - ciphertext size indep. of hierarchy depth (unknown from LWE)
- ⇒ adaptive security: sec. degrades exp. in hierarchy depth

Some pairing-based HIBEs

- **GS-HIBE** [GS'03]: $\text{BDH} \Rightarrow \text{IND-IDCPA}$ (in RO model)
- **BB-HIBE** [BB'04]: $\text{BDDH} \Rightarrow \text{IND-sIDCPA}$
- **BW-HIBE** [BW'05]: $\text{2-DLIN} \Rightarrow \text{anonIND-sIDCPA}$
- **BBG-HIBE** [BBG'05]: $d\text{-BDDH} \Rightarrow \text{IND-sIDCPA}$
ciphertext size **indep.** of hierarchy depth (unknown from LWE)
- **DualSys-HIBE** [LW'10]: (various, short) $\Rightarrow \text{IND-IDCPA}$
same size as BBG and good for poly. depth hierarchies

Example: BBG-HIBE [BBG'05]

- $S(\lambda)$: $(G, G_T, g, q) \leftarrow \text{GenBilGroup}(\lambda), \alpha \leftarrow F_p$
 $pp := [g, y \leftarrow g^\alpha, g_2, g_3, h_1, \dots, h_d] \in G \quad ; \quad mk := (g_2)^\alpha$
- $K(mk, (id_1, \dots, id_k))$:
 $sk \leftarrow [mk \cdot (h_1^{id_1} \dots h_k^{id_k} g_3)^r, g^r, h_{k+1}^r, h_{k+2}^r \dots, h_d^r]$


Example: BBG-HIBE [BBG'05]

- S(λ): $(G, G_T, g, q) \leftarrow \text{GenBilGroup}(\lambda)$, $\alpha \leftarrow F_p$
 $pp := [g, y \leftarrow g^\alpha, g_2, g_3, h_1, \dots, h_d] \in G$; $mk := (g_2)^\alpha$
- Delegation:** $sk(id_1, \dots, id_k) \rightarrow sk(id_1, \dots, id_k, id_{k+1})$
 $sk \leftarrow [mk \cdot (h_1^{id_1} \dots h_k^{id_k} g_3)^r, g^r, h_{k+1}^r, h_{k+2}^r \dots, h_d^r]$


absorb and re-randomize

Example: BBG-HIBE [BBG'05]

- S(λ): $(G, G_T, g, q) \leftarrow \text{GenBilGroup}(\lambda)$, $\alpha \leftarrow F_p$
 $pp := [g, y \leftarrow g^\alpha, g_2, g_3, h_1, \dots, h_d] \in G$; $mk := (g_2)^\alpha$
- Delegation:** $sk(id_1, \dots, id_k) \rightarrow sk(id_1, \dots, id_k, id_{k+1})$
 $sk \leftarrow [mk \cdot (h_1^{id_1} \dots h_k^{id_k} g_3)^r, g^r, h_{k+1}^r, h_{k+2}^r, \dots, h_d^r]$

 $sk \leftarrow [mk \cdot (h_1^{id_1} \dots h_{k+1}^{id_{k+1}} g_3)^t, g^t, h_{k+2}^t, \dots, h_d^t]$

Example: BBG-HIBE [BBG'05]

- $S(\lambda)$: $(G, G_T, g, q) \leftarrow \text{GenBilGroup}(\lambda)$, $\alpha \leftarrow F_p$
 $pp := [g, y \leftarrow g^\alpha, g_2, g_3, h_1, \dots, h_d] \in G$; $mk := (g_2)^\alpha$
- $E(pp, (\text{id}_1, \dots, \text{id}_k), m)$: $s \leftarrow F_p$ and do
 $C \leftarrow (g^s, (h_1^{\text{id}_1} \cdots h_k^{\text{id}_k} g_3)^s, m \cdot e(y, g_2)^s)$

Final note: many further generalizations

- Wildcard IBE [ABCD...’06]

encrypt to: $ID = (id_1, id_2, *, id_3, *, id_4)$

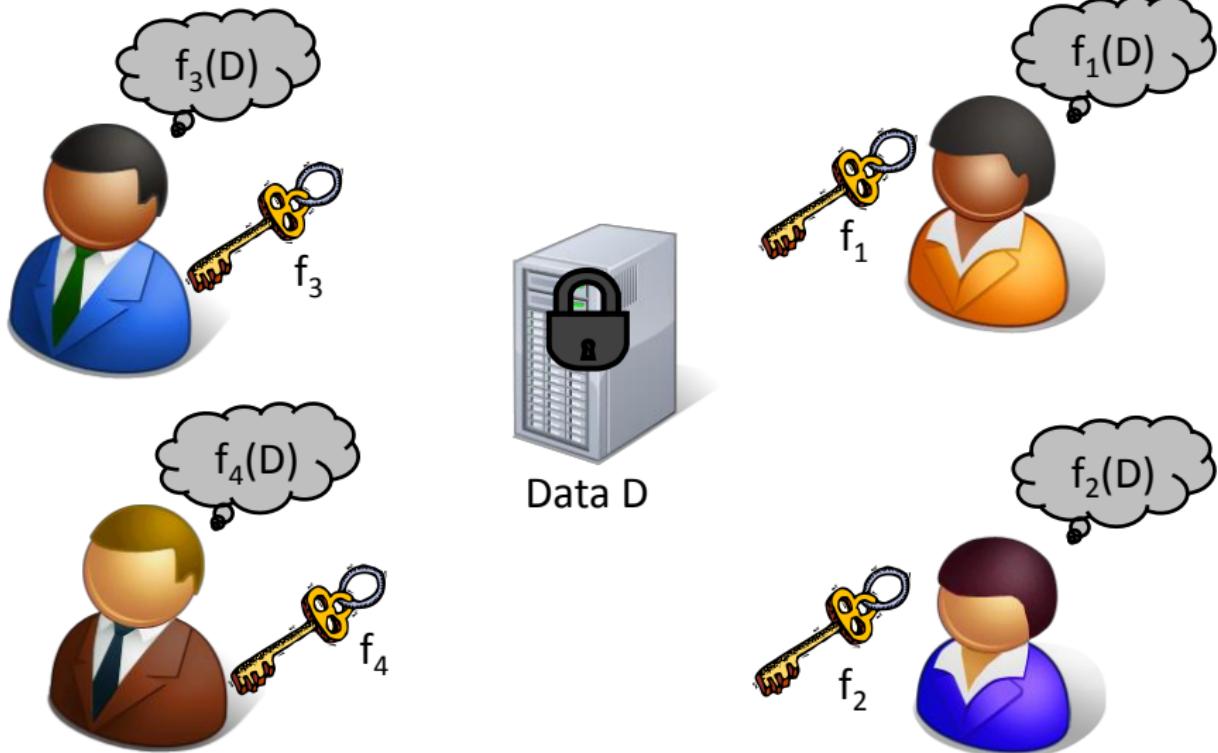
- Hidden vector encryption [BW’06] and
Inner product encryption [KSW’08]
 - Support more general searches on encrypted data
e.g. range queries, conjunctive queries, ...
- Next topic: attribute based encryption [SW’05]

Functional Encryption

Motivation

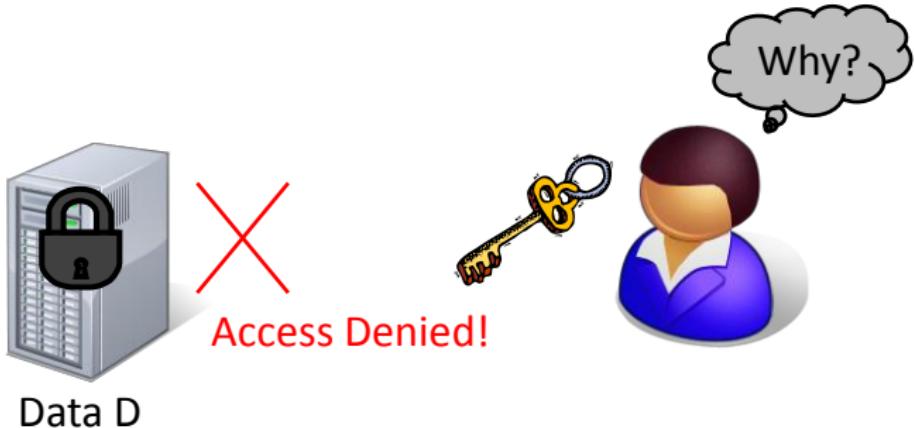


A Broad Vision



A First Step

How might we hide the access policy itself?



Inner Product Encryption

λ = security parameter

n = vector length

Setup(λ, n):

generate public parameters PP and master key MSK

KeyGen(\vec{v} , MSK):

generate a user key for a given vector of length n

Encrypt(PP, M, \vec{x}):

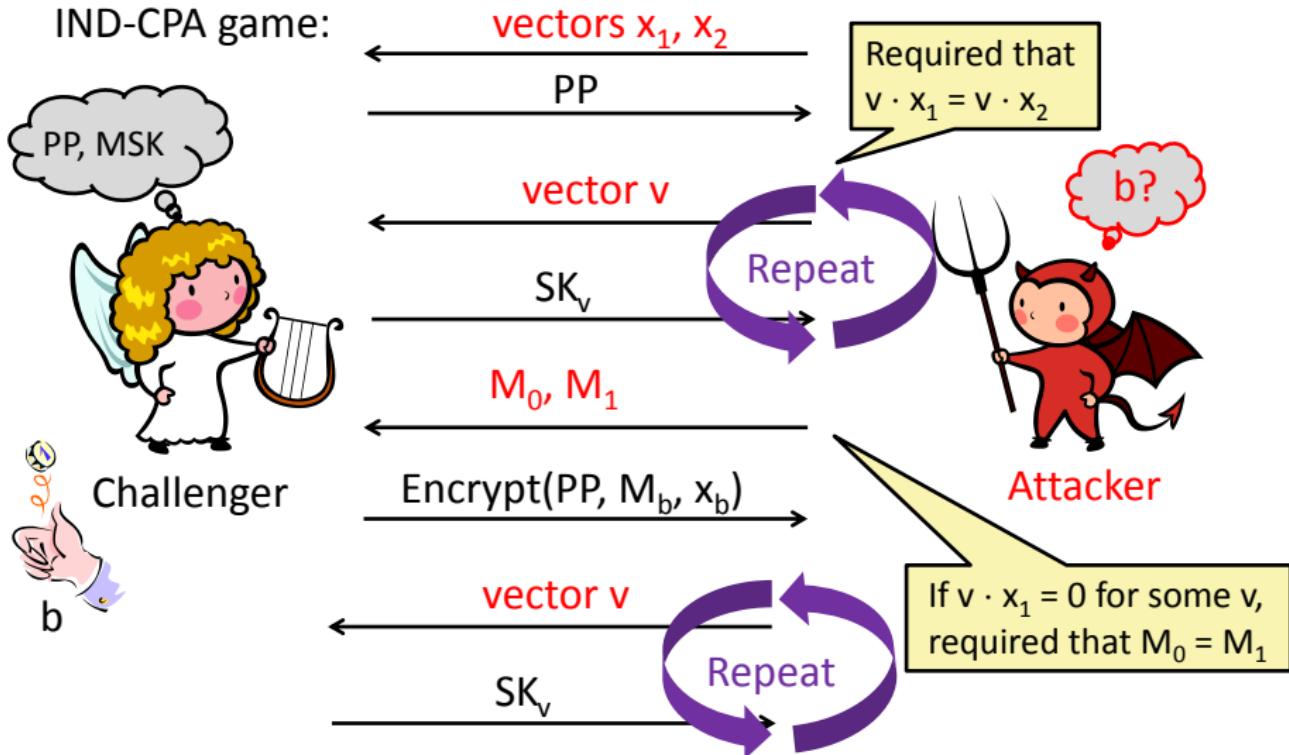
encrypt message M under a vector of length n

Decrypt(CT, SK):

decrypt ciphertext using a key: successful iff $\vec{x} \cdot \vec{v} \equiv 0$

Security Definition for IPE (selective) [KSW08]

IND-CPA game:



Construction Intuition

We want to compute $\vec{x} \cdot \vec{v}$ while hiding \vec{x} :

Basic idea: compute $\vec{x} \cdot \vec{v}$ in the exponent

CT: $\circlearrowleft g^{x_1} \circlearrowright g^{x_2} g^{x_3} \dots g^{x_n}$

SK: $\circlearrowleft g^{v_1} \circlearrowright g^{v_2} g^{v_3} \dots g^{v_n}$

$$\frac{e(g, g)^{x_1 v_1} e(g, g)^{x_2 v_2} e(g, g)}$$

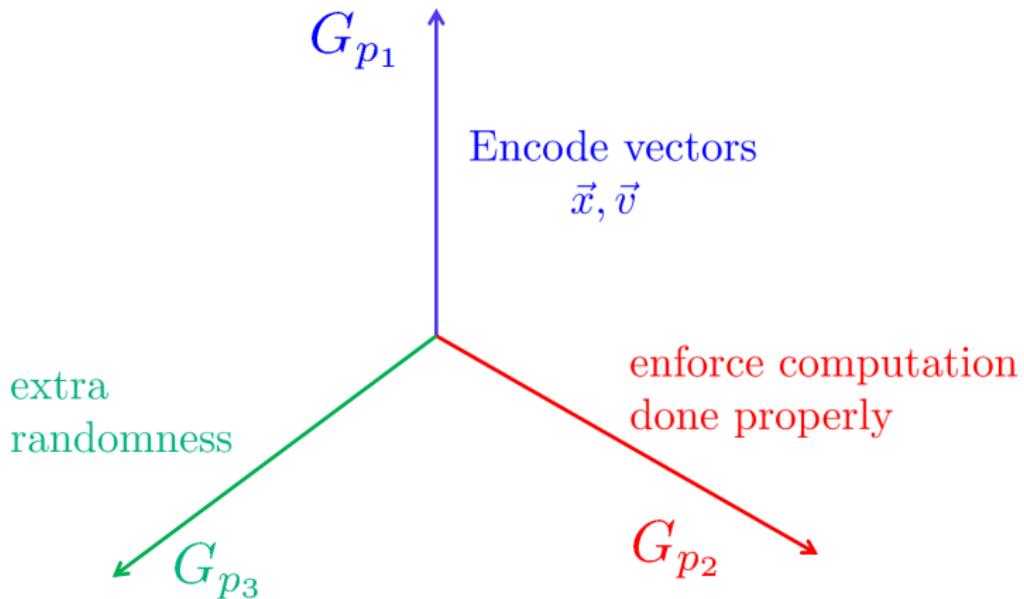
add randomness that
cancels only when
computation done properly

Some remaining problems:

Can tell if $x_i = 0$ or not

Can permute the computation

Subgroup Roles



IPE Construction (Predicate Only) [KSW08]

Setup (λ, n) : generate G of order $N = p_1 p_2 p_3$

$$PP = \textcolor{red}{g}, \textcolor{teal}{g}, \textcolor{blue}{gR}, \{\textcolor{red}{h_i}V_i, \textcolor{red}{k_i}W_i\}_{i=1}^n$$

Encrypt($\vec{x} = (x_1, x_2, \dots, x_n)$):

choose $s, \alpha, \beta \in \mathbb{Z}_N$

2 parallel systems,
stayed tuned for why
we want this

$$\text{CT} = \textcolor{red}{g}^s, \{(h_i V_i)^s (\textcolor{blue}{g} V_i)^{\alpha x_i} U_i, (\textcolor{red}{k}_i W_i)^s (\textcolor{blue}{g} V_i)^{\beta x_i} Z_i\}_{i=1}^n$$

KeyGen($\vec{v} = (v_1, v_2, \dots, v_n)$):

$$\text{SK} = \textcolor{teal}{AB} \prod_{i=1}^n h_i^{-r_i} k_i^{-t_i}, \{g^{r_i} g^{\delta v_i}, g^{t_i} g^{\sigma v_i}\}_{i=1}^n$$

Decryption

CT:

$$g^s$$

$$(h_i V_i)^s (g V_i)^{\alpha x_i} U_i$$

$$(k_i W_i)^s (g V_i)^{\beta x_i} Z_i$$

Take product for $i = 1$ to n

SK:

$$AB \prod_{i=1}^n h_i^{-r_i} k_i^{-t_i}$$

$$g^{r_i} g^{\delta v_i}$$

$$g^{t_i} g^{\sigma v_i}$$

$$\frac{\prod_{i=1}^n e(g, h_i)^{-sr_i} e(g, k_i)^{-st_i}}{e(g, h_i)^{sr_i} e(g, g)^{\alpha \delta x_i v_i}} \frac{}{e(g, k_i)^{st_i} e(g, g)^{\beta \sigma x_i v_i}}$$

$$e(g, g)^{(\alpha \delta + \beta \sigma) \vec{x} \cdot \vec{v}}$$

= 1 if and only if $\vec{x} \cdot \vec{v} \equiv 0 \pmod{p_1}$

Proof Intuition

New challenge:

Adversary attempting to distinguish CT under \vec{x} from CT under \vec{y} requests key for \vec{v} such that $\vec{v} \cdot \vec{x} = \vec{v} \cdot \vec{y} = 0$

Natural approach would be a hybrid changing \vec{x} to \vec{y} one coordinate at a time:

$$(x_1, \dots, x_n) \Rightarrow (x_1, \dots, x_i, y_{i+1}, \dots, y_n) \Rightarrow (y_1, \dots, y_n)$$

may not be orthogonal to \vec{v} !

Proof Intuition

Idea: use two parallel systems, change one half at a time

Hybrid structure: CT exponent vectors change as

$$(\vec{x}, \vec{x}) \Rightarrow (\vec{x}, \vec{0}) \Rightarrow (\vec{x}, \vec{y}) \Rightarrow (\vec{0}, \vec{y}) \Rightarrow (\vec{y}, \vec{y})$$



Why go through $\vec{0}$?

$\vec{0}$ is orthogonal to everything, and we can go from \vec{x} to $\vec{0}$ with Subgroup Decision Assumptions



Back to General Functionalities...

A general definition [BSW11]:

Def. 1.

A functionality F is a function $F : K \times X \rightarrow \{0, 1\}^*$ whose domain is the product of a key space K and a plaintext space X .

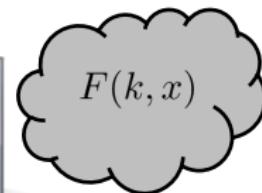
It is required that K include an “empty key” ϵ .



$k \in K$



$x \in X$



Formal Specification

$\lambda = \text{security parameter}$

Setup(λ):

generate public parameters PP and master key MSK

KeyGen(k , MSK):

generate a user key for a given $k \in K$

Encrypt(PP, $x \in X$):

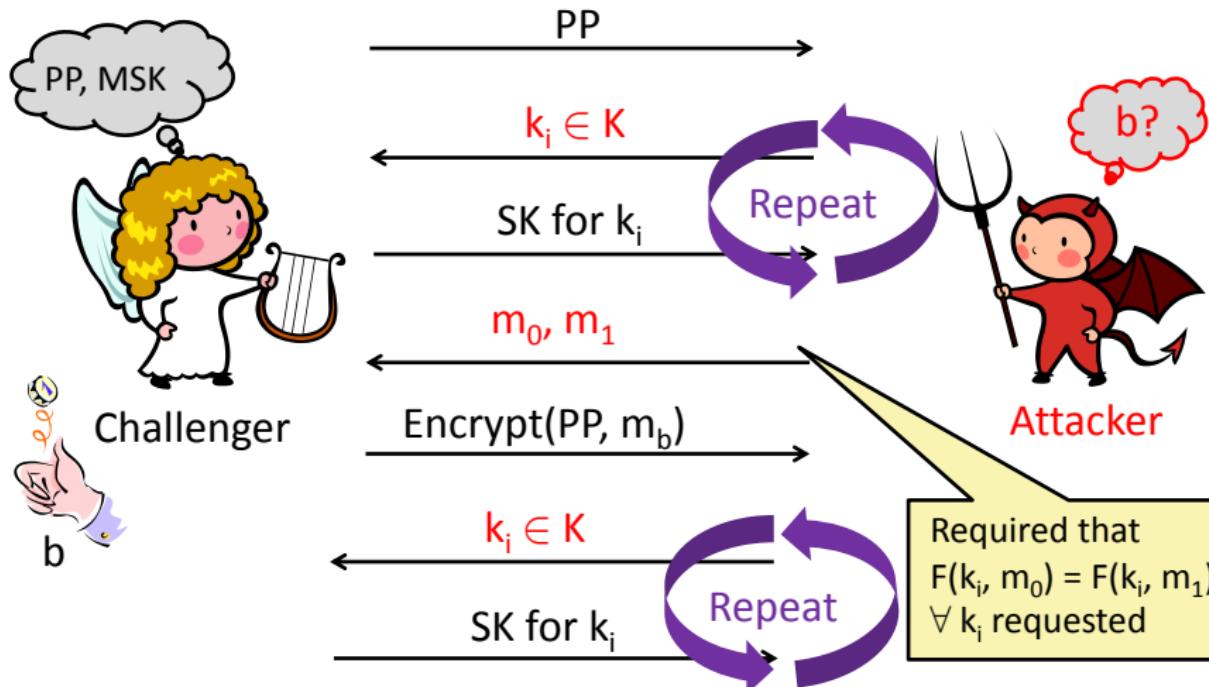
encrypt message x

Decrypt(CT, SK):

decrypt ciphertext using a key to obtain $F(k, x)$

IND Game-Based Security Definition

IND-CPA game:



When IND Security May Be Insufficient

It does not capture computational properties of the functionality F :

Example:

Consider $K = \{\epsilon\}$, $X = \{0, 1\}^n$

$$F(\epsilon, x) := \pi(x)$$

one-way permutation

Proposed Construction:

$$\text{Encrypt}(x) = x$$

$$F(\epsilon, x_1) = F(\epsilon, x_2) \Leftrightarrow x_1 = x_2$$

So this is “secure” under game-based definition!

A Further Example

from [O10]: Let $\mathcal{F} = \{f_1, \dots, f_n\}$ be set of functions associated with keys

Let g be a function so that given $f_1(x) \parallel \dots \parallel f_n(x)$,
it is hard to guess $g(x)$

And $g(x) = g(y) \Leftrightarrow f_1(x) \parallel \dots \parallel f_n(x) = f_1(y) \parallel \dots \parallel f_n(y)$

Let $(\text{Setup}, \text{Encrypt}, \text{Decrypt})$ be a Public Key Encryption scheme

Let $(\text{Setup}^*, \text{KeyGen}^*, \text{Encrypt}^*, \text{Decrypt}^*)$ be a FE scheme

New FE scheme:

Setup: run Setup and Setup^* to get $(pk, sk), (pk^*, sk^*)$

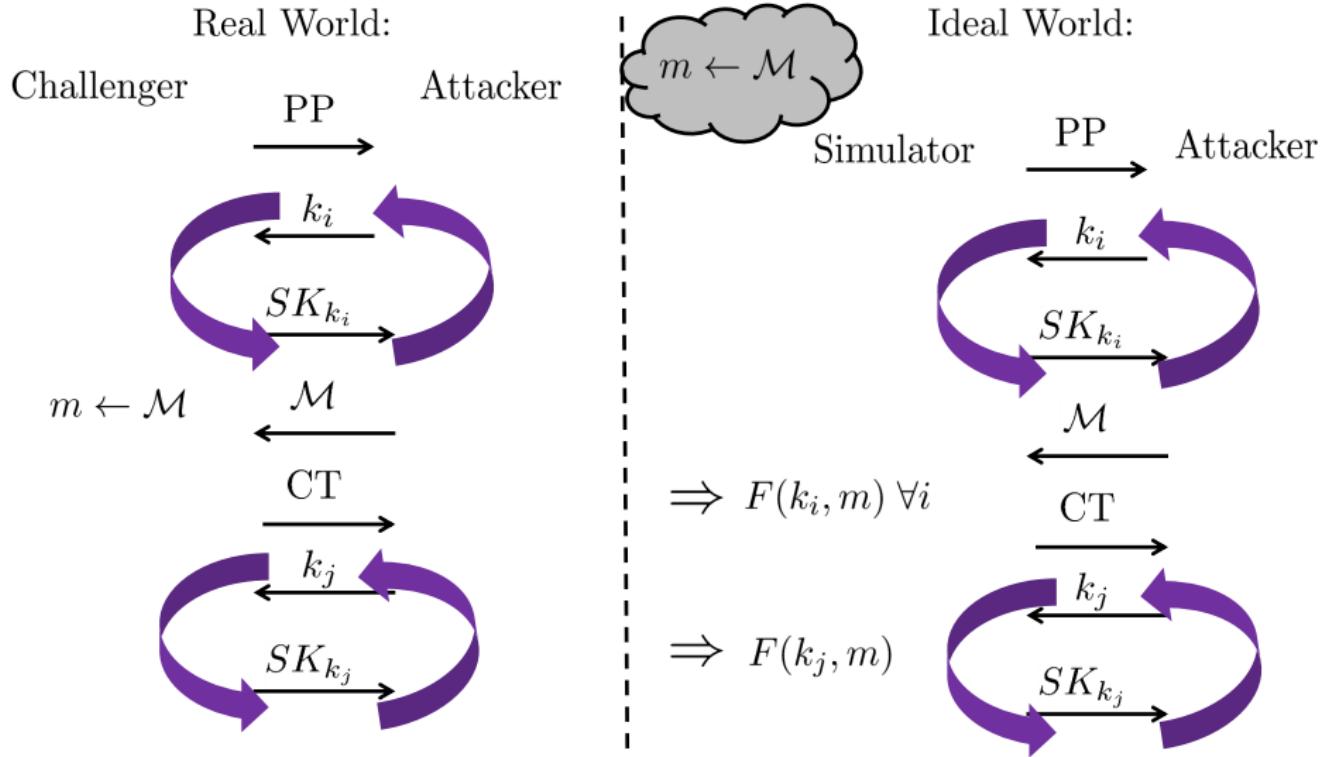
secret share sk as $\omega_1, \dots, \omega_n$

set $pk := pk \parallel pk^*, sk := \omega_1 \parallel \dots \parallel \omega_n \parallel sk^*$

KeyGen(f_i): run $\text{KeyGen}^*(f_i)$ to get sk_i , set $sk_i := \omega_i \parallel sk_i$

Encrypt(m): run $\text{Encrypt}^*(m)$ and $\text{Encrypt}(g(m))$, concat results

A Simulation-Based Security Definition

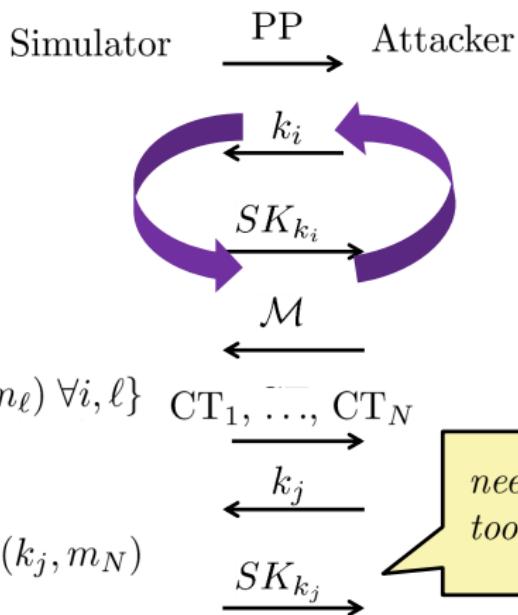


Impossibility Results for Sim-Based Security

$m_1, m_2 \leftarrow \mathcal{M} \leftarrow \mathcal{M}$

Ideal World:

[BSW11]



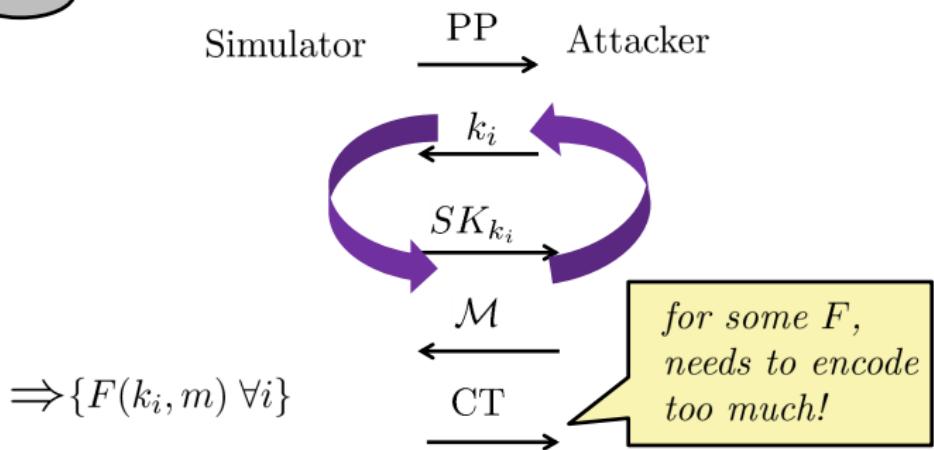
$$\Rightarrow F(k_j, m_1), \dots, F(k_j, m_N)$$

Impossibility Results for Sim-Based Security

$m \leftarrow \mathcal{M}$

Ideal World:

[AGVW12]



$$\Rightarrow \{F(k_i, m) \ \forall i\}$$

*Can avoid this by bounding the queries

Homomorphic Encryption

Computing on Encrypted Data

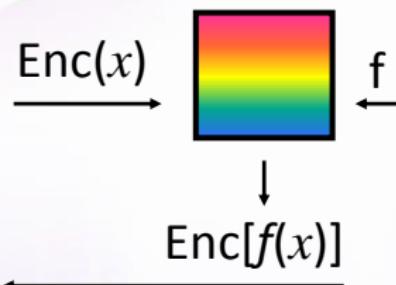
I want to delegate processing of my data,
without giving away access to it.

Outsourcing Computation

“I want to delegate the computation to the cloud,
but the cloud shouldn’t see my input”



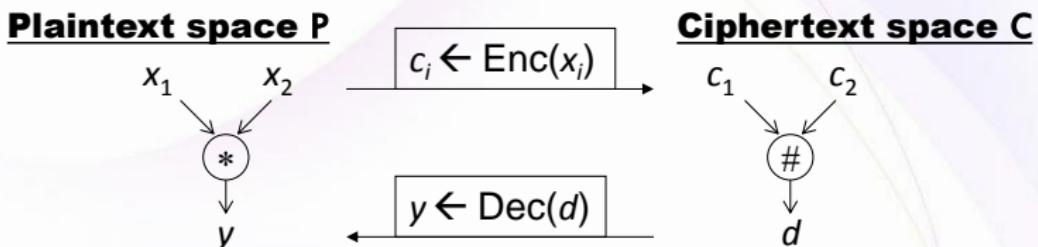
Client
(Input: x)



Server/Cloud
(Function: f)

Privacy Homomorphisms

- Rivest-Adelman-Dertouzos 1978



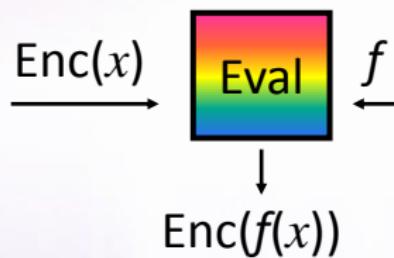
Example: RSA_encrypt_(e,N)(x) = $x^e \bmod N$

• $x_1^e \times x_2^e = (x_1 \times x_2)^e \bmod N$

“Somewhat Homomorphic”: can compute some functions on encrypted data, but not all

“Fully Homomorphic” Encryption

- Encryption for which we can compute **arbitrary functions** on the encrypted data



Some Notations

- An encryption scheme: (KeyGen, Enc, Dec)
 - Plaintext-space = {0,1}
 - $(pk, sk) \leftarrow \text{KeyGen}(\$)$, $c \leftarrow \text{Enc}_{pk}(b)$, $b \leftarrow \text{Dec}_{sk}(c)$
- Semantic security [GM'84]:
 $(pk, \text{Enc}_{pk}(0)) \approx (pk, \text{Enc}_{pk}(1))$
≈ means indistinguishable by efficient algorithms

Homomorphic Encryption (HE)

- $H = \{\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval}\}$
- $c^* \leftarrow \text{Eval}_{pk}(f, c)$
- Homomorphic: $\text{Dec}_{sk}(\text{Eval}_{pk}(f, \text{Enc}_{pk}(x))) = f(x)$
 - c^* may not look like a “fresh” ciphertext
 - As long as it decrypts to $f(x)$
- Compact: Decrypting c^* easier than computing f
 - Otherwise we could use $\text{Eval}_{pk}(f, c) = (f, c)$ and $\text{Dec}_{sk}(f, c) = f(\text{Dec}_{sk}(c))$
 - Technically, $|c^*|$ independent of the complexity of f

Fully Homomorphic Encryption

- ➊ First plausible candidate in [Gen'09]
 - ➌ Security from hard problems in ideal lattices
 - ➌ Polynomially slower than computing in the clear
 - ➌ Big polynomial though
- ➋ Many advances since
 - ➌ Other hardness assumptions
 - ➌ LWE, RLWE, NTRU, approximate-GCD
 - ➌ More efficient
 - ➌ Other “Advanced properties”
 - ➌ Multi-key, Identity-based, ...

This Talk

- ➊ Regev-like somewhat-homomorphic encryption
 - ➌ Adding homomorphism to [Reg'05] cryptosystem
 - ➍ Security based on LWE, Ring-LWE
 - ➎ Based on [BV'11, BGV'12, B'12]
- ➋ Bootstrapping to get FHE [Gen'09]
- ➌ Packed ciphertexts for efficiency
 - ➎ Based on [SV'11, BGV'12, GHS'12]
- ➍ Not in this talk: a new LWE-based scheme
 - ➏ [Gentry-Sahai-Waters CRYPTO 2013]

Learning with Errors [Reg'05]

Many equivalent forms, this is one of them:

- Parameters: q (modulus), n (dimension)
- Secret: a random short vector $\mathbf{s} \in Z_q^n$
- Input: many pairs (\mathbf{a}_i, b_i)
 - $\mathbf{a}_i \in Z_q^n$ is random, $b_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i$ ($mod\ q$)
 - e_i is short
- Goal: find the secret \mathbf{s}
 - Or distinguish (\mathbf{a}_i, b_i) from random in Z_q^{n+1}

[Regev'05, Peikert'09]: As hard as some worst-case lattice problems in dim n (for certain range of params)

Regev's Cryptosystem [Reg'05]

- The shared-key variant (enough for us)
- Secret key: vector s' , denote $\mathbf{s} = (\mathbf{s}', \mathbf{1})$
- Encrypt($\sigma \in \{0,1\}$)
 - $c = (a, b)$ s.t. $b = \sigma \frac{q}{2} - \langle s', a \rangle + e \pmod{q}$
 - Convenient to write $\langle \mathbf{s}, \mathbf{c} \rangle = \sigma \frac{q}{2} + e \pmod{q}$
- Decrypt(\mathbf{s}, \mathbf{c})
 - Output 0 if $|\langle \mathbf{s}, \mathbf{c} \rangle \bmod q| \leq q/4$, else output 1
 - Correct decryption as long as error $< q/4$

Security: If LWE is hard, ciphertext is pseudorandom

Additive Homomorphism

- If $\langle s, c_i \rangle \approx \sigma_i \frac{q}{2} \pmod{q}$ then
 $\langle s, c_1 + c_2 \rangle \approx (\sigma_1 \oplus \sigma_2) \frac{q}{2} \pmod{q}$
- Error doubles on addition
- Correct decryption as long as the error $< q/4$

How to Multiply [BV'11, B'12]

• Step 1: Tensor Product

- If $\langle s, c_i \rangle \approx \sigma_i \frac{q}{2} \pmod{q}$ and s is small ($|s| \ll q$)
then $\langle s \otimes s, c_1 \otimes c_2 \rangle \approx \sigma_1 \sigma_2 \frac{q^2}{4} \pmod{q^2}$
 - Error has extra additive terms of size $\approx |s| \cdot q \ll q^2$
- So $c^* = \text{round}((c_1 \otimes c_2) / \frac{q}{2})$ encrypts $\sigma_1 \sigma_2$ relative to secret key $s^* = (s \otimes s)$
 - Rounding adds another small additive error
- But the dimension squares on multiply

How to Multiply [BV'11, B'12]

Step 2: Dimension Reduction

- Publish “key-switching gadget” to translate c^* wrt s^* $\rightarrow c$ wrt s
 - Essentially an encryption of s^* under s
- $n \times n^2$ rational matrix W s.t. $s^T \times W \approx s^* \pmod{q}$
- Given c^* , compute $c \leftarrow \text{Round}(W \times c^*) \pmod{q}$
- $\langle s, c \rangle \approx s^T \times W \times c^* \approx \langle s^*, c^* \rangle \approx \sigma \frac{q}{2} \pmod{q}$
 - Some extra work to keep error from growing too much
 - Still secure under reasonable hardness assumptions

Somewhat Homomorphic Encryption

- Error doubles on addition, grows by $\text{poly}(n)$ factor on multiplication (e.g., n^2 factor)
 - When computing a depth- d circuit we have $|\text{output-error}| \leq |\text{input-error}| \cdot n^{2d}$
- Setting parameters:
 - Start from $|\text{input-error}| \leq n^d$ (say)
 - Set $q > 4n^d \cdot n^{2d} = 4n^{3d}$
 - Set the dimension large enough to get security
- $|\text{output-error}| < q/4$, so no decryption errors

FHE via Bootstrapping [Gen'09]

- So far, circuits of pre-determined depth

x_1

x_2

...

x_t



$C(x_1, x_2, \dots, x_t)$

FHE via Bootstrapping [Gen'09]

- So far, circuits of pre-determined depth



- Can eval $y = C(x_1, x_2, \dots, x_n)$ when x_i 's are “fresh”
- But y is an “evaluated ciphertext”
 - Can still be decrypted
 - But eval $C'(y)$ will increase noise too much

FHE via Bootstrapping [Gen'09]

- So far, circuits of pre-determined depth

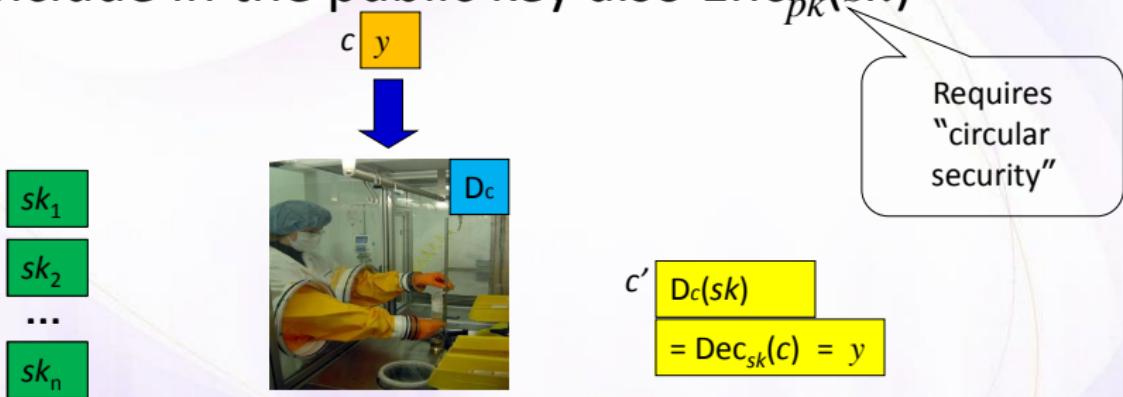


- Bootstrapping to handle deeper circuits

- We have a noisy evaluated ciphertext y
- Want to get another y with less noise

FHE via Bootstrapping [Gen'09]

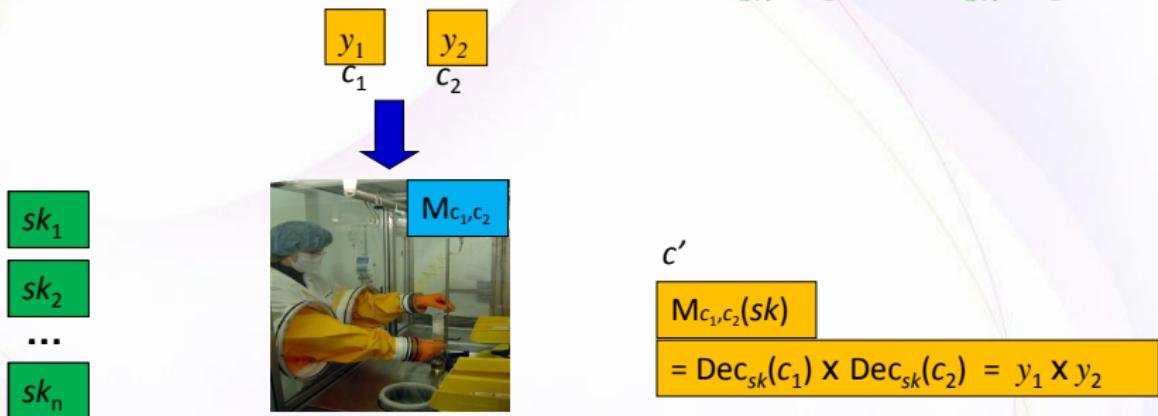
- For ciphertext c , consider $\mathbf{D}_c(sk) = \text{Dec}_{sk}(c)$
 - Hope: $D_c(*)$ is a low-depth circuit (on input sk)
- Include in the public key also $\text{Enc}_{pk}(sk)$



- Homomorphic computation applied only to the “fresh” encryption of sk

FHE via Bootstrapping [Gen'09]

- Similarly define $\mathbf{M}_{c_1, c_2}(sk) = \text{Dec}_{sk}(c_1) \cdot \text{Dec}_{sk}(c_1)$



- Homomorphic computation applied only to the “fresh” encryption of sk

(In)Efficiency of This Scheme

- The LWE-based somewhat-homomorphic scheme has depth- $\tilde{O}(\log qn)$ decryption circuit
- To get FHE need modulus $q \geq 2^{polylog(k)}$ and dimension $n \geq \tilde{\Omega}(k)$
 - k is the security parameter
- The ciphertext-size is $\tilde{\Omega}(k)$ bits
- Key-switching matrix is of size $\tilde{\Omega}(k^3)$ bits
 - Each multiplication takes at least $\tilde{\Omega}(k^3)$ times
 - $\tilde{\Omega}(k^3)$ slowdown vs. computing in the clear

Better Efficiency with Ring-LWE

- Replace \mathbb{Z} by $\mathbb{Z}[X]/F(X)$
 - F is a degree- d polynomial with $d = \tilde{\Theta}(k)$
 - Can get security with lower dimension
 - $n = \tilde{\Theta}(k/d)$, as low as $n = 2$
 - The ciphertext-size still $\tilde{\Omega}(k)$ bits
 - But key-switching matrix size only $\tilde{\Theta}(k)$ bits
 - It includes $n^2 \times n = 8$ ring elements
- $\tilde{\Theta}(k)$ slowdown vs. computing in the clear

Ciphertext Packing

- Cannot reduce ciphertext size below $\tilde{\Theta}(k)$
- But we can pack more bits in each ciphertext
- Recall decryption: $ptxt \leftarrow MSB(\langle s, c \rangle \bmod q)$
 - $ptxt$ is a polynomial in $R_2 = \mathbb{Z}[X]/(F(X), 2)$
- Use cyclotomic rings, $F(X) = \Phi_m(X)$
- Use CRT in R_2 to pack many bits inside m
 - The cryptosystem remains unchanged
 - Encoding/decoding of bits inside plaintext polys

Plaintext Algebra

- $\Phi_m(X)$ irreducible over \mathbb{Z} , but not mod 2
 - $\Phi_m(X) \equiv \prod_{j=1}^{\ell} F_j(X) \pmod{2}$
 - F_j 's are irreducible, all have the same degree d
 - degree d is the order of 2 in Z_m^*
 - For some m 's we can get $\ell = \frac{\phi(m)}{d} = \Omega\left(\frac{m}{\log m}\right)$
- $R_2 = \mathbb{Z}_2[X]/\Phi_m$ is a direct sum, $R_2 = \bigoplus_j R_{2,j}$
 - $R_{2,j} = \mathbb{Z}_2[X]/F_j(X) \cong GF(2^d)$
- 1-1 mapping $a \in R_2 \leftrightarrow [\alpha_1, \dots, \alpha_\ell] \in GF(2^d)^\ell$

Plaintext Slots

- Plaintext $a \in R_2$ encodes ℓ values $\alpha_j \in GF(2^d)$
 - To embed plaintext bits, use $a_j \in GF(2) \subset GF(2^d)$
- Ops $+, \times$ in R_2 work independently on the slots
 - ℓ -ADD: $a + a' \cong [\alpha_1 + \alpha'_1, \dots, \alpha_\ell + \alpha'_\ell]$
 - ℓ -MUL: $a \times a' \cong [\alpha_1 \times \alpha'_1, \dots, \alpha_\ell \times \alpha'_\ell]$
- If $\ell = \tilde{\Omega}(k)$ then our $\tilde{\Theta}(k)$ -bit ciphertext can hold $\tilde{\Omega}(k)$ plaintext bits
 - Ciphertext-expansion ratio only $\text{polylog}(k)$

Aside: an ℓ -SELECT Operation

$$\begin{array}{c} x \\ \hline \end{array} \begin{array}{|c|c|c|c|c|c|c|} \hline & \textcolor{red}{x_1} & \textcolor{yellow}{x_2} & \textcolor{yellow}{x_3} & \textcolor{red}{x_4} & \textcolor{blue}{x_5} & \textcolor{red}{x_6} & \textcolor{blue}{x_7} \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|c|} \hline & \textcolor{red}{x_1} & 0 & 0 & \textcolor{red}{x_4} & 0 & \textcolor{red}{x_6} & 0 \\ \hline \end{array}$$

+

$$\begin{array}{c} x \\ \hline \end{array} \begin{array}{|c|c|c|c|c|c|c|} \hline & \textcolor{blue}{x_8} & \textcolor{red}{x_9} & \textcolor{red}{x_{10}} & \textcolor{yellow}{x_{11}} & \textcolor{red}{x_{12}} & \textcolor{blue}{x_{13}} & \textcolor{red}{x_{14}} \\ \hline 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|c|} \hline & 0 & \textcolor{red}{x_9} & \textcolor{red}{x_{10}} & 0 & \textcolor{red}{x_{12}} & 0 & \textcolor{red}{x_{14}} \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline & \textcolor{red}{x_1} & \textcolor{red}{x_9} & \textcolor{red}{x_{10}} & \textcolor{red}{x_4} & \textcolor{red}{x_{12}} & \textcolor{red}{x_6} & \textcolor{red}{x_{14}} \\ \hline \end{array}$$

- We will use this later

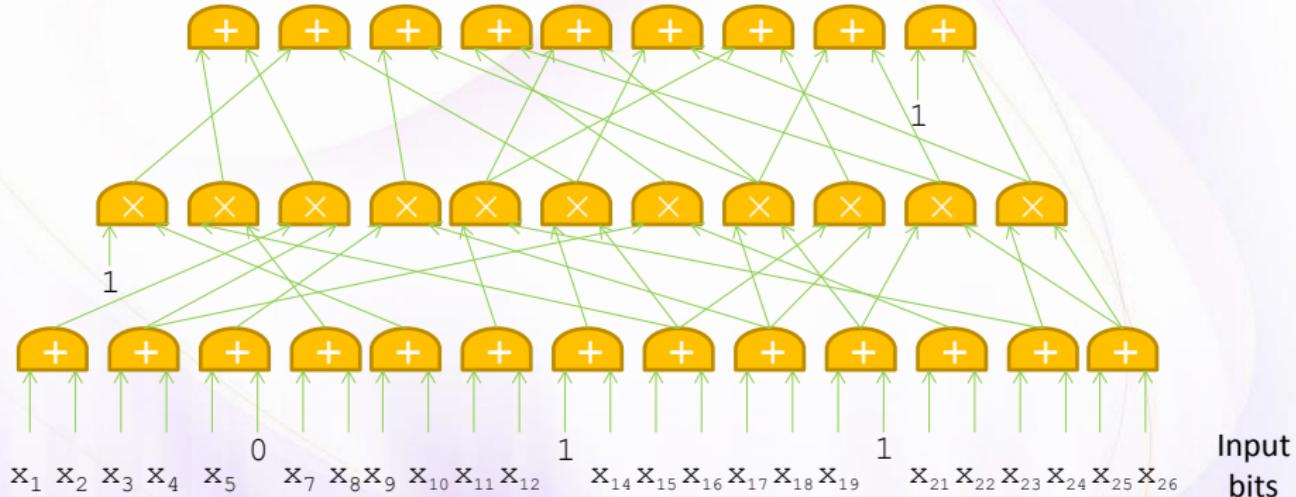
Homomorphic SIMD [SV'11]

- SIMD = Single Instruction Multiple Data
- Computing the same function on ℓ inputs at the price of one computation
 - Overhead only $\text{polylog}(k)$
- Pack the inputs into the slots
 - Bit-slice, inputs to j 'th instance go in j 'th slots
- Compute the function once
- After decryption, decode the ℓ output bits from the output plaintext polynomial

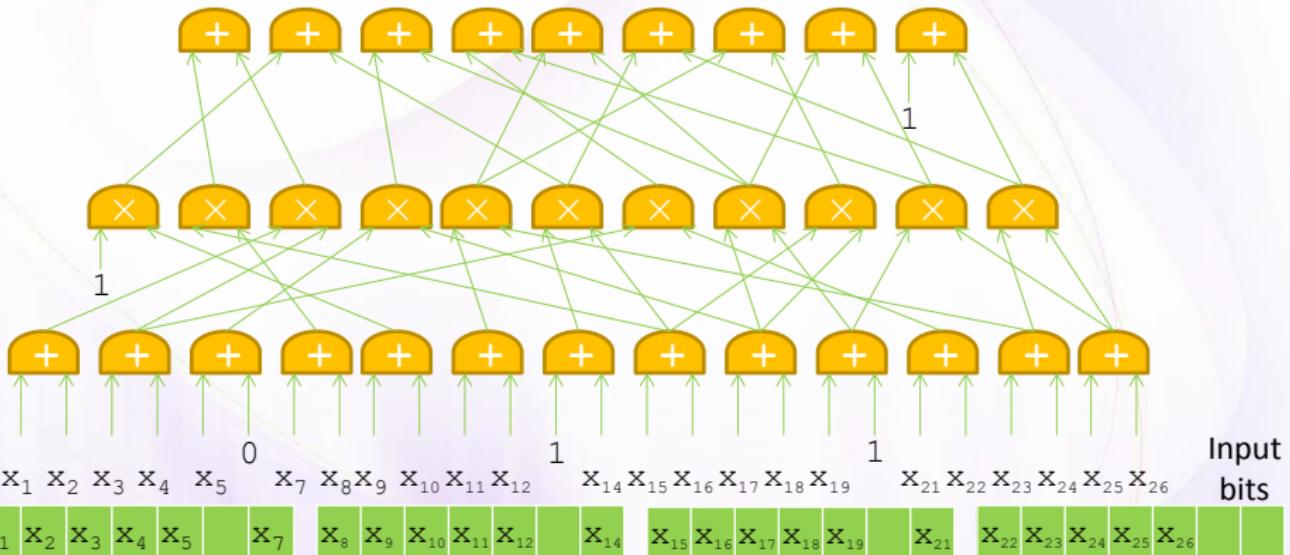
Beyond SIMD Computation

- ➊ To reduce overhead for a single computation:
 - ➌ Pack all input bits in just a few ciphertexts
 - ➌ Compute while keeping everything packed
- ➋ How to do this?

So you want to compute some function...



So you want to compute some function using SIMD...



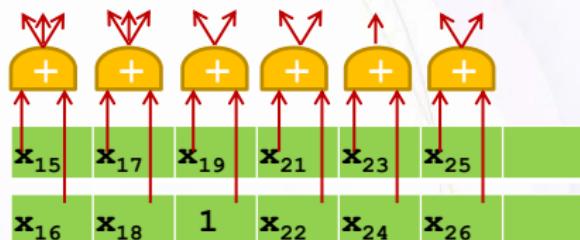
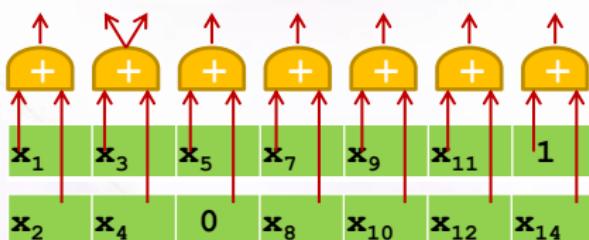
Routing Values Between Levels

- We need to map this

x_1	x_2	x_3	x_4	x_5	0	x_7	
x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	1	x_{21}	

x_8	x_9	x_{10}	x_{11}	x_{12}	1	x_{14}	
x_{22}	x_{23}	x_{24}	x_{25}	x_{26}			

- Into that ... so we can use ℓ -add



- Is there a natural operation on polynomials that moves values between slots?

Using Automorphisms

- The operation $\kappa_t: a(X) \mapsto a(X^t) \in R_2$
- Under some conditions on m , exists $t \in Z_m^*$ s.t.,
 - For any $a \in R_2$ encoding $a \leftrightarrow [\alpha_1, \alpha_2, \dots, \alpha_\ell]$,
 $\kappa_t(a) \leftrightarrow [\alpha_2, \dots, \alpha_\ell, \alpha_1]$
 - t is a generator of $Z_m^*/(2)$ (if it exists)
- Once we have rotations, we can get every permutation on the plaintext slots
 - Using only $O(\log \ell)$ shifts and SELECTs [GHS'12]
- How to implement κ_t homomorphically?

Homomorphic Automorphism

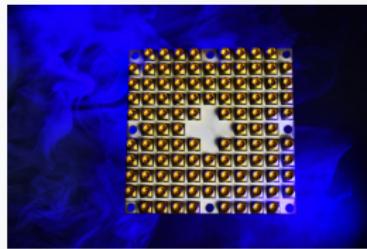
- Recall decryption via inner product $\langle \mathbf{s}, \mathbf{c} \rangle \in R_q$
 - If $a(X) = \langle \mathbf{s}(X), \mathbf{c}(X) \rangle \text{ mod } (\Phi_m(X), q)$ then also
 $a(X^t) = \langle \mathbf{s}(X^t), \mathbf{c}(X^t) \rangle \text{ mod } (\Phi_m(X^t), q)$
 - Since $\Phi_m(X) | \Phi_m(X^t)$ for any $t \in Z_m^*$, then also
 $a(X^t) = \langle \mathbf{s}(X^t), \mathbf{c}(X^t) \rangle \text{ mod } (\Phi_m(X), q)$
- Therefore $\mathbf{c}' = \kappa_t(\mathbf{c})$ is an encryption of
 $a' = \kappa_t(a)$ relative to key $\mathbf{s}' = \kappa_t(\mathbf{s})$
- Can publish key-switching matrix $W[\mathbf{s}' \rightarrow \mathbf{s}]$ to
get back an encryption relative to \mathbf{s}

Summary of RLWE HE encryption

- Native plaintext space $R_2 = \mathbb{Z}_2[X]/\Phi_m$
 - $a \in R_2$ used to pack ℓ values $\alpha_j \in GF(2^d)$
- sk is $s \in R_q$, ctxt is a pair $(c_0, c_1) \in R_q^2$
- Decryption is $a := MSB(\langle (c_0, c_1), (s, 1) \rangle)$
 - Inner product over R_q
- Homomorphic addition, multiplication work element-size on the α_j 's
- Homomorphic automorphism to move α_j 's between the slots

Post Quantum Cryptography

The Rise of Quantum Computing



Intel's 49-qubit chip
"Tangle-Lake"
January 2018

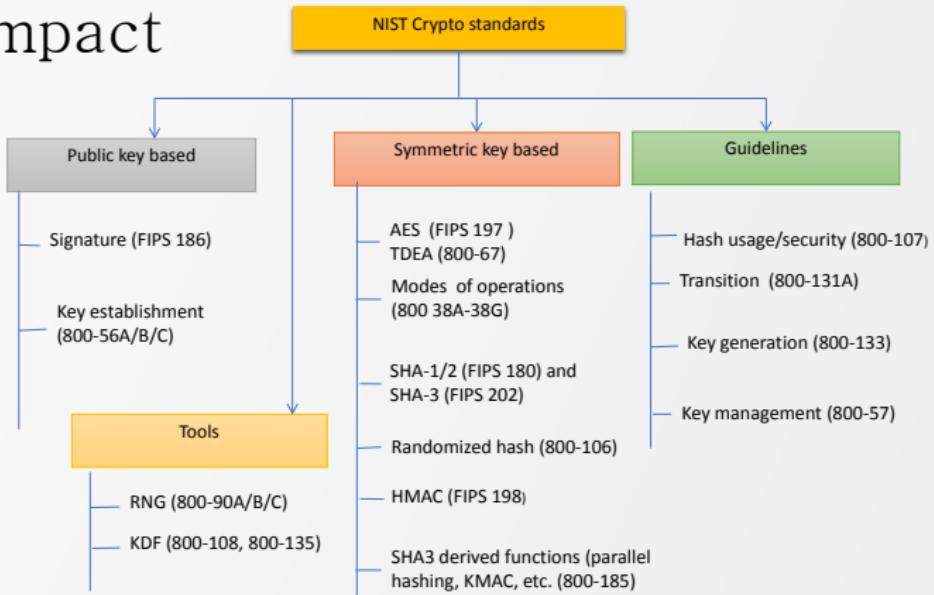


Google's 72-qubit chip
"Bristlecone"
March 2018

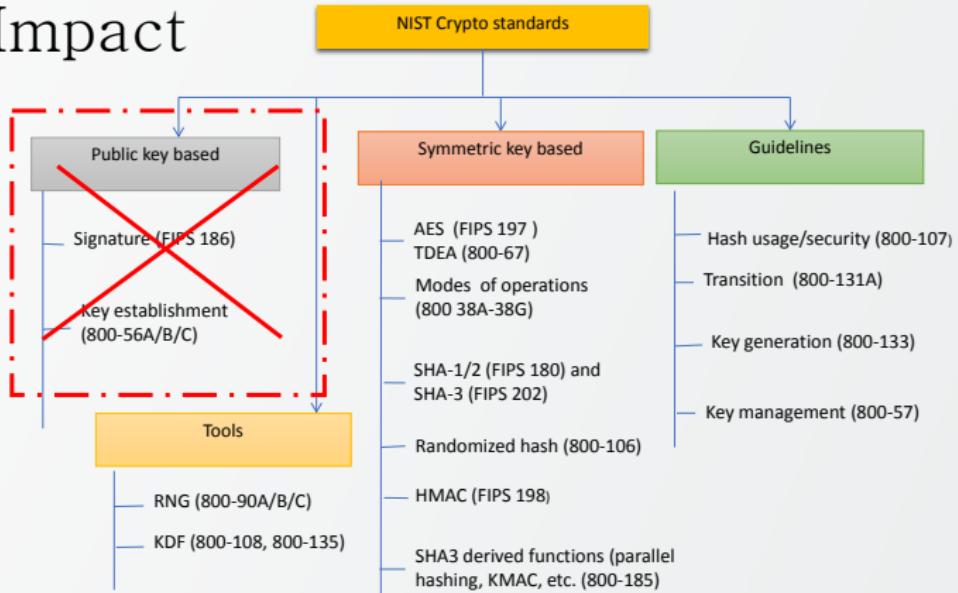


IBM's 50-qubit
quantum computer
November 2017

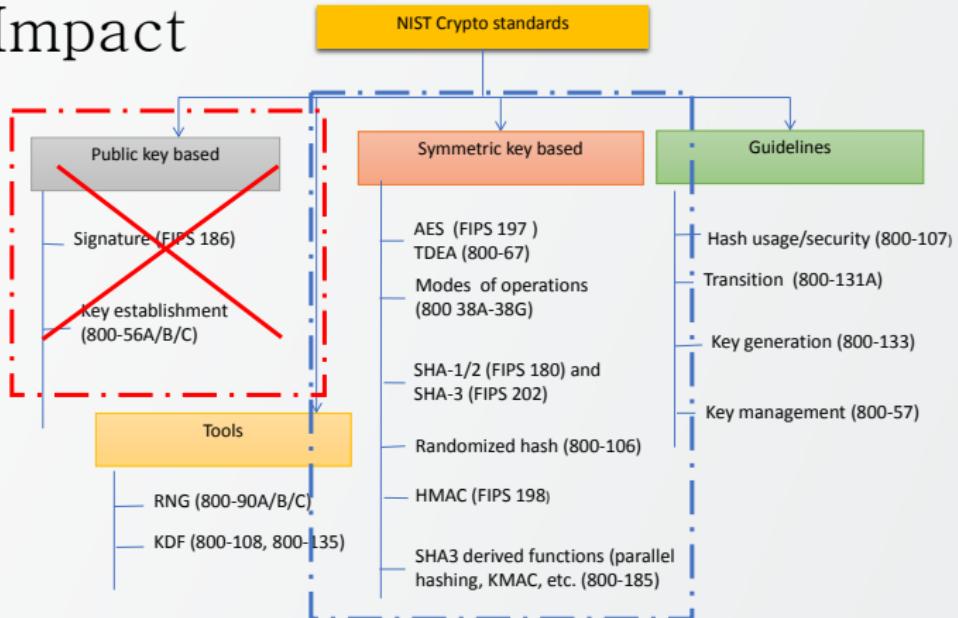
The Impact



The Impact

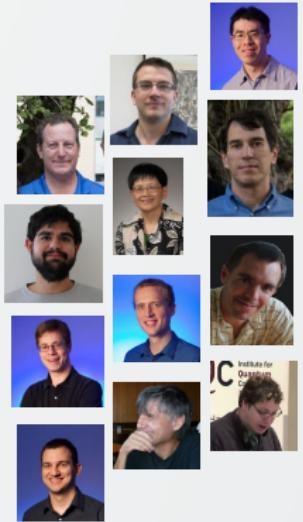


The Impact



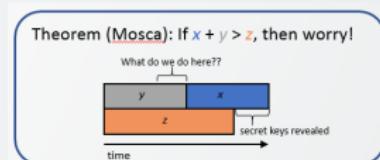
The NIST PQC Project

- 2009 – NIST publishes a PQC survey
 - [Quantum Resistant Public Key Cryptography: A Survey](#) [D. Cooper, R. Perlner]
- 2012 – NIST begins PQC project
 - Research and build team
 - Work with other standards organizations (ETSI, IETF, ISO/IEC SC 27)
- April 2015 – 1st NIST PQC Workshop



PQC Standardization – when?

- There had been much debate about whether it is too early to look into PQC standardization
- When will a (large-scale) quantum computer be built?
 - “There is a 1 in 7 chance that some fundamental public-key crypto will be broken by quantum by 2026, and a 1 in 2 chance of the same by 2031.”
 - Dr. Michele Mosca, (April 2015)



- Our experience tells us that we need (at least) several years to develop and deploy PQC standards

The Decision to Move Forward

- Aug 2015 – NSA statement
 - ... “IAD will initiate a transition to *quantum resistant algorithms* in the **not too distant future** ...”
- Feb 2016 – NIST Report on PQC ([NISTIR 8105](#))
- Feb 2016 – NIST announcement at PQCrypto
- We see our role as **managing a process** of achieving community consensus in a **transparent** and **timely** manner
- We do not expect to “pick a winner”

Timeline

- Aug 2016 – Draft submission requirements & evaluation criteria
- Dec 2016 – Final requirements and criteria
- Nov 2017 – Deadline for submissions
- Apr 2018 – NIST PQC Workshop – submitters' presentations
- 2018/2019 – 2nd Round begins (smaller number of submissions)
 - minor changes allowed
- Aug 2019 – 2nd NIST PQC Workshop
- 2020/2021 - Select algorithms or start a 3rd Round
- 2022-2024 - Draft standards available
- NIST will release reports on progress and selection rationale

Scope

- **Signatures**

- Public-key schemes for generating/verifying signatures (see FIPS 186-4)

- **Encryption**

- Key transport from one party to another
- Exchanging encrypted secret values between two parties to establish shared secret value (see SP 800-56B)

- **Key-establishment (KEMs)**

- Schemes like Diffie-Hellman key exchange (see SP 800-56A)

Differences with past Competitions

- Post-quantum cryptography is more complicated than AES/SHA-3
 - No silver bullet - each candidate has some disadvantage
 - Not enough research on quantum algorithms to ensure confidence for some schemes
- We do not expect to select just one algorithm
 - Ideally, several algorithms will emerge as "good choices"
- We will narrow our focus at some point
 - This does not mean algorithms are "out"
- Requirements/timeline could potentially change based on developments in the field

Complexities

- Much broader scope – three crypto primitives
- Both classical and quantum attacks
 - Security strength assessment on specific parameter selections
- Consider various theoretical security models and practical attacks
 - Provably security vs. security against instantiation or implementation related security flaws and pitfalls
- Multiple tradeoff factors
 - Security, performance, key size, signature size, side-channel resistance countermeasures
- Migrations into new and existing applications
 - TLS, IKE, code signing, PKI infrastructure, and much more
- Not exactly a competition – it is and it isn't

The Selection Criteria

- **Security** - against both classical and quantum attacks
- **Performance** - measured on various "classical" platforms
- **Other properties**
 - Drop-in replacements - Compatibility with existing protocols and networks
 - Perfect forward secrecy
 - Resistance to side-channel attacks
 - Simplicity and flexibility
 - Misuse resistance, and
 - More

Security Analysis

- Security definitions (proofs recommended, but not required)
 - IND-CPA/IND-CCA2 for encryption, KEMS
 - EUF-CMA for signatures
 - Used to judge whether an attack is relevant
- Quantum/classical algorithm complexity
 - Classical computers may have the cheapest attacks in practice
 - Stability of best known attack complexity
 - Precise security claim against quantum computation
- Quality and quantity of prior cryptanalysis

Quantum Security

- No clear consensus on best way to measure quantum attacks
- Uncertainties
 - The possibility that new quantum algorithms will be discovered, leading to new attacks
 - The performance characteristics of future quantum computers, such as their cost, speed and memory size
- For PQC standardization, need to specify concrete parameters with security estimates

Security Strength Categories

Level	Security Description
I	At least as hard to break as AES128 (exhaustive key search)
II	At least as hard to break as SHA256 (collision search)
III	At least as hard to break as AES192 (exhaustive key search)
IV	At least as hard to break as SHA384 (collision search)
V	At least as hard to break as AES256 (exhaustive key search)

- Computational resources should be measured using a variety of metrics
- NIST asked submitters to focus on levels 1,2, and 3
 - Levels 4 and 5 for high security
- These are understood to be preliminary estimates

Cost and Performance

- Standardized post-quantum cryptography will be implemented in "classical" platforms
- Ideally, implementable on wide variety of platforms and applications
- May need to standardize [more than one](#) algorithm for each function to accommodate different application environments
- Allowing parallel implementation for improving efficiency is certainly a plus
- **Preliminary conclusions:** efficiency likely OK, but key sizes may pose a significant challenge

Complexities – Part 2

- Assess classical security
 - Many PQC schemes are relatively new. It'll take years to understand their classical security. Let alone quantum security.
- We need to deal with new situations which we haven't considered before, e.g.
 - Decryption failure
 - State management for hash based signatures
 - Public-key encryption vs. key-exchange issues
 - Public-key encryption IND-CCA2
 - Ephemeral key exchange (no key-pair reuse, consider passive attacks, IND-CPA)
 - Auxiliary functions/algorithms, e.g.
 - Gaussian distribution sampling/simulation

Intellectual Property

- "NIST does not object in principle to algorithms or implementations which may require the use of a patent claim, where technical reasons justify this approach, but will consider any factors which could hinder adoption in the evaluation process."
 - All submitters must declare known patents
 - Reminder: submitters turn in your signed IP statements
- Submissions (and implementations) are freely available for public review and evaluation
- In Round 1, all submissions should be evaluated on their technical merits.

Submissions

- 37 preliminary submissions (early deadline Sep 2017)
- 82 total submissions received
 - 69 accepted as “complete and proper” (5 since withdrawn)

	Signatures	KEM/Encryption	Overall
Lattice-based	5	21	26
Code-based	2	17	19
Multi-variate	7	2	9
Symmetric/Hash-based	3		3
Other	2	5	7
Total	19	45	64

Numbers

- We have a total of 278 submitters
 - 67 of those were on more than one submission
 - Distribution: [212, 30, 22, 7, 2, 1, 4, 1]
- Most submissions cover security levels 1,3, and 5.
 - 10 submissions target only the lower levels 1,2,3
 - CFPKM, CompactLWE, Emblem/R.Emblem, NTRU-HRSS-KEM, PQRSA Enc/Sig, QC MDPC-KEM, Gravity-SPHINCS, HiMQ-3, RaCoSS
 - 6 submissions target only the high security levels 4,5
 - Classic McEliece, GuessAgain, Hila5, Mersenne-756839, NTRUprime, KCL

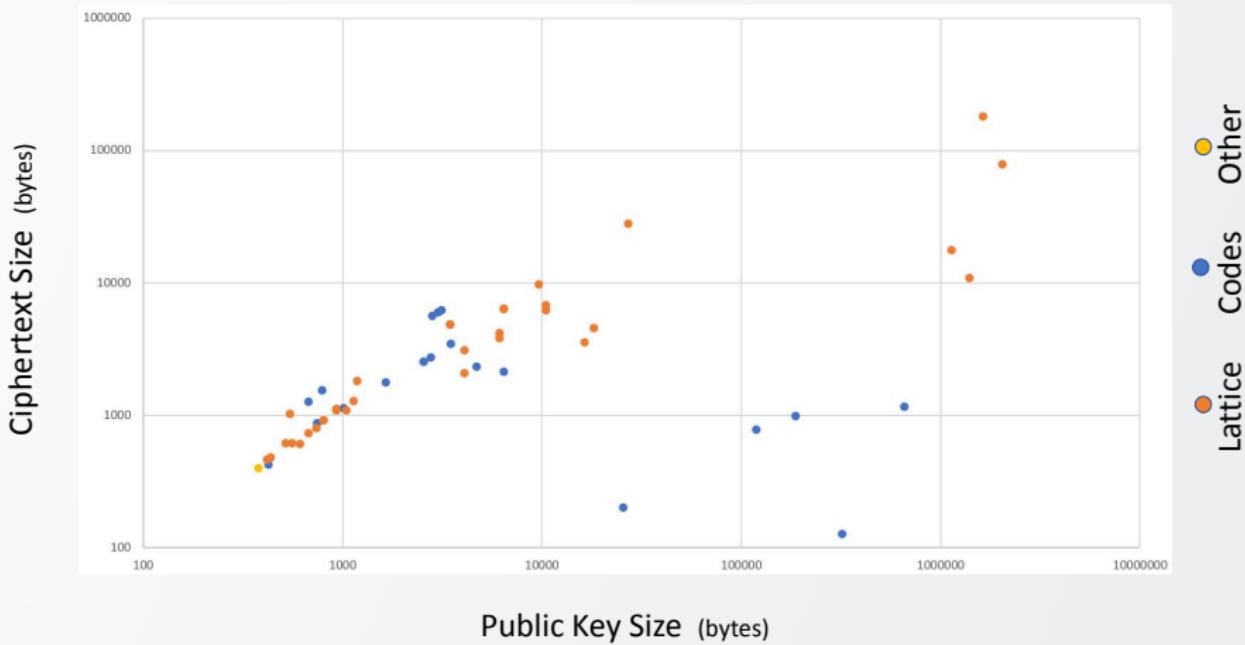
25 Countries, 16 States, 6 Continents



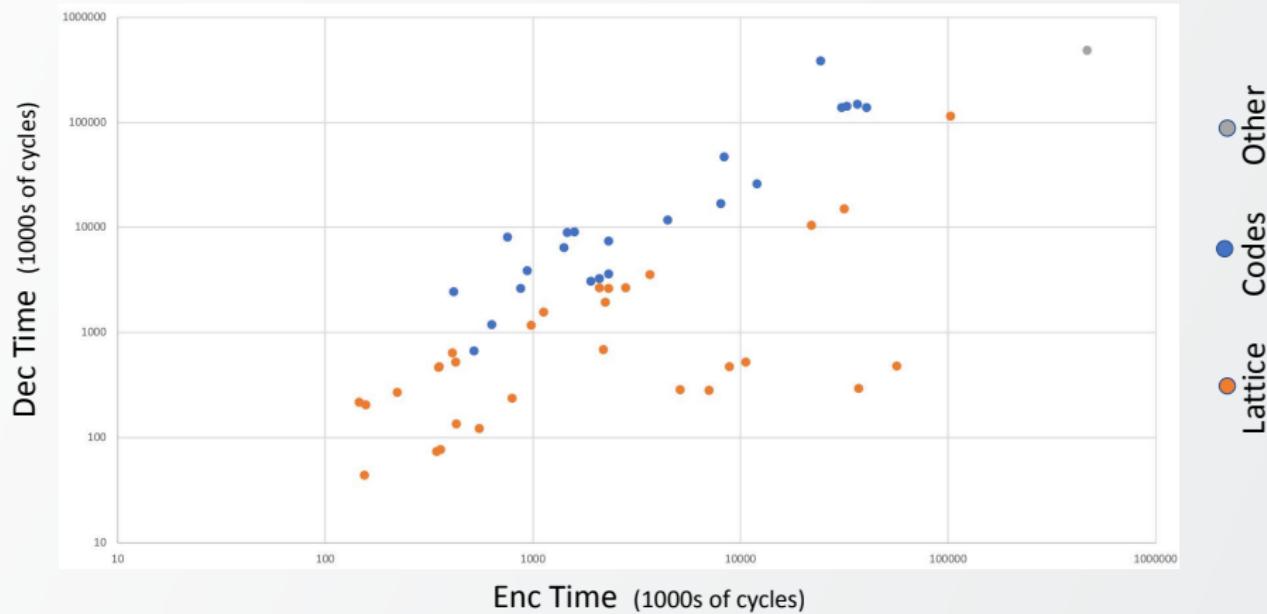
Key Sizes & Performance Graphs

- Reminder: “It is important to note that performance considerations will **NOT** play a major role in the early portion of the evaluation process.”
- Disclaimer – These are from the optimized implementations submitted to us. We know better implementations exist/will exist.
- These charts should mainly be used to see general patterns
 - While performance will vary with implementations, key sizes won’t

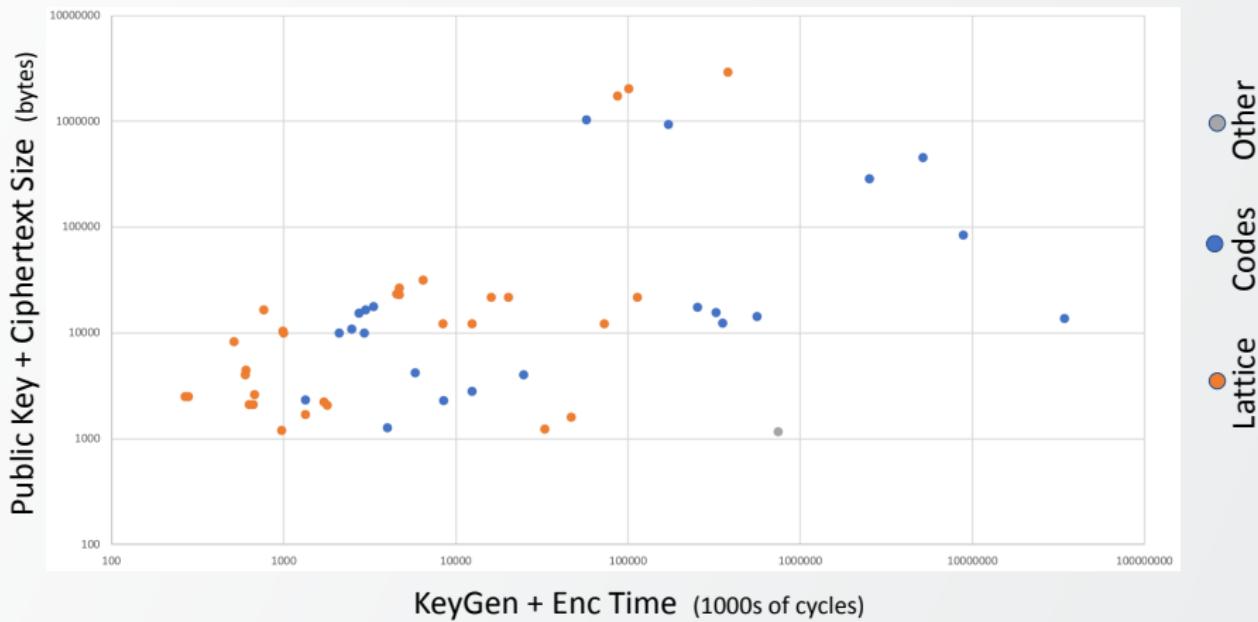
KEM/Encryption (Category 1)



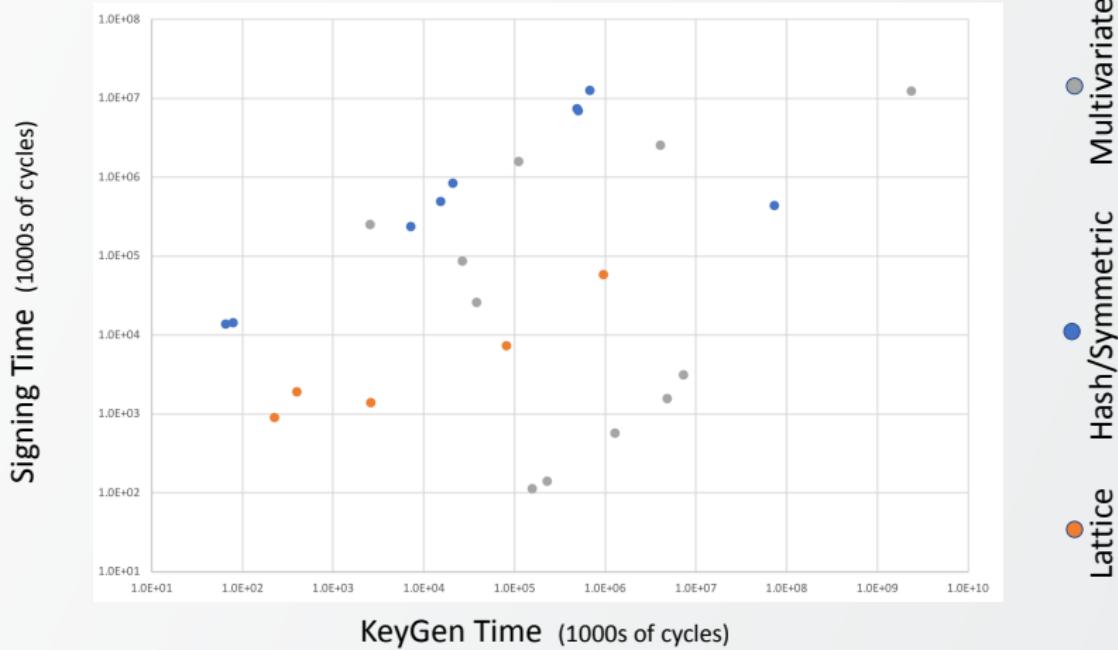
KEM/Encryption (Category 3) Performance



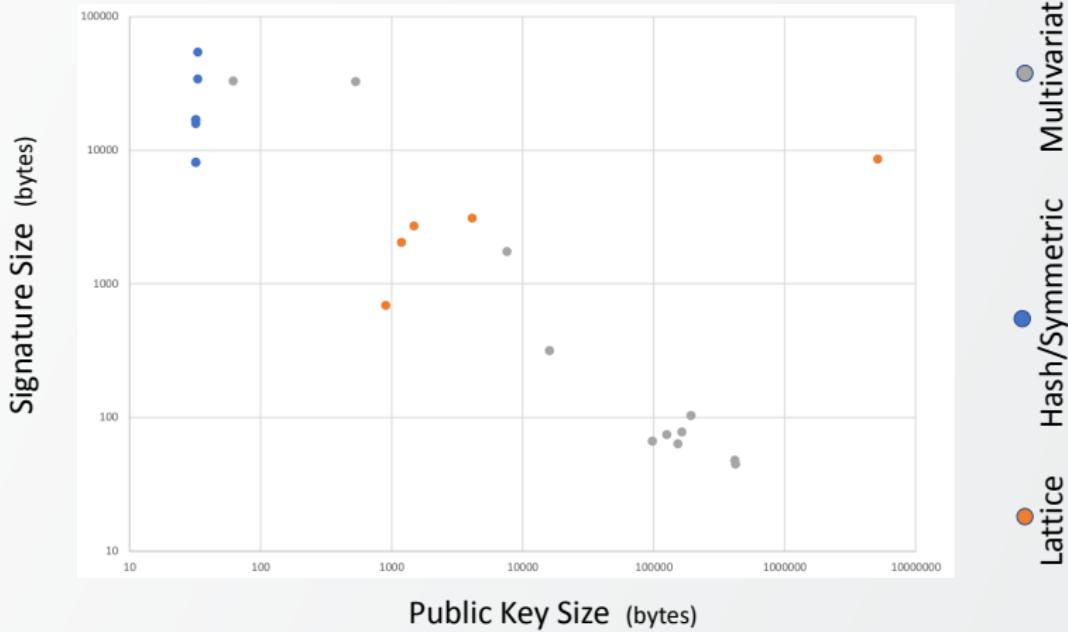
KEM/Encryption (Category 3) Performance by Size



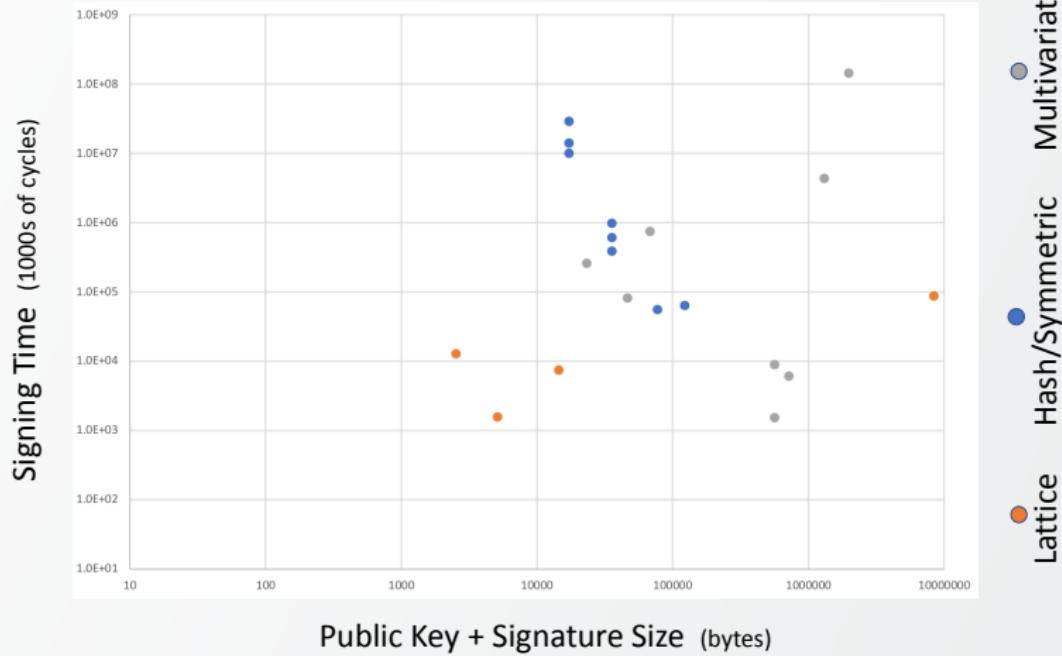
Signatures (Category 1) Performance



Signature (Category 1) Sizes



Signatures (Category 3) Performance by Size



Discussion and Questions

- Since the draft call for proposals was announced, the NIST team has actively interacted with submitters and researchers
- The questions include
 - APIs to support different ancillary functions
 - Using third party libraries
 - Submission format
 - Decryption failure
 - etc.
- The topics discussed at pqc-forum@nist.gov include
 - Quantum vs. classical security strength
 - Security notions (IND-CCA2, IND-CPA, etc.)
 - Random number generation
 - Key exchange vs. key encapsulation (KEMs)
 - Implementation details, (constant-time, etc.....)
 - Official comments on submissions
 - IP/patent issues
- Answers to the common questions and summaries on the major discussion topics are added to the FAQ at www.nist.gov/pqcrypto

Official Comments

- Submit “official comments” on our website using link for each submissions
 - Alternatively, post in the pqc-forum with “Official Comment: NameOfSubmission” in the subject line
- Comments can be minor (bug fixes) or major (breaks)
 - Often are questions, which are answered by submitters
- 38 submissions have received official comments
 - 26 submissions have none
 - 18 submissions have 2 or less
- 210 official comments so far
 - ~60% of these are on 10 submissions.

Transition and Migration

- NIST will update guidance when PQC standards are available
- A “hybrid mode” has been proposed as a transition/migration step towards PQC
 - Such a mode combines a classical algorithm with a post-quantum one
 - Current FIPS 140 validation will only validate the NIST-approved (classical) component
 - The PQC standardization will only consider the post-quantum component
- NIST plans to consider (stateful) hash-based signatures as an early candidates for standardization
 - Only for specific applications like code signing
 - We hope to hear from industry and implementers on the urgency/impact of hash-based signatures

Standards Organizations

- We are aware that many standards organizations and expert groups are working on PQC
 - IEEE P1363.3 has standardized some lattice-based schemes
 - IETF is taking action in specifying stateful hash-based signatures
 - ETSI has released quantum-safe cryptography reports
 - EU expert groups PQCRYPTO and SAFEcrypto made recommendations and released reports
 - ISO/IEC JTC 1 SC27 has already had a 2 year study period for quantum-resistant cryptography and is developing a standing document (SD)
- NIST is interacting and collaborating with these organizations and groups

What's Next?

- 2nd NIST PQC Standardization Workshop, Aug 2019
- Sometime before then, we will pick a smaller number of submissions that we feel are the most promising
 - For these, tweaks are allowed in the 2nd Round
- Will be announced on pqc-forum (and our webpage)
- If not selected:
 - Might be eliminated from the standardization process
 - Or might be kept for future consideration, but not in 2nd Round

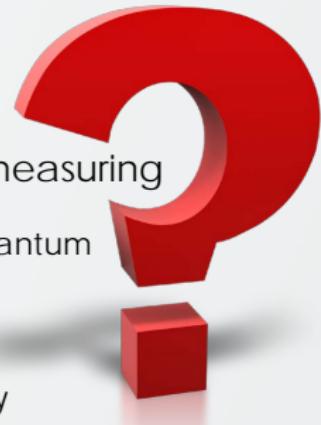
What does NIST want from you?

- Continue to analyze the submissions
 - Publish and present your work
- Implementations for a variety of platforms
- See how these will fit into applications/protocols
 - Dig into the details – is there anything different from current practice (such as the way to use auxiliary functions)
- Participate in the pqc-forum
- Send us your questions/feedback:



pqc-comments@nist.gov

Questions we have…



- Does NIST need to provide more guidance on measuring the complexity of quantum attacks?
 - Should we specify one or two plausible models of quantum computers?
- Or on complexity of classical attacks?
 - how to deal with attacks with extremely high memory
- How should we handle submissions which are very similar?
 - Keep one? Keep both? Merge them? How?
- What constitutes unacceptable key sizes or performance?

Summary

- Post-quantum crypto standardization will be a long journey
- We have seen many complexities, and know more lie ahead
- Be prepared to transition to new algorithms in 10 years
- We will continue to work in an **open and transparent** manner with the crypto community for PQC standards
- Check out www.nist.gov/pqcrypto
 - Sign up for the pqc-forum for announcements & discussion

