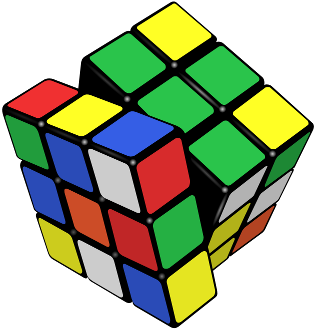
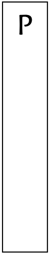
Cryptography in Cyclic Groups (episode 3)



Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of **prime** order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where

$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

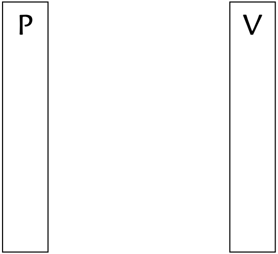
P sends $s = k + cx \mod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

P sends $s = k + cx \mod q$

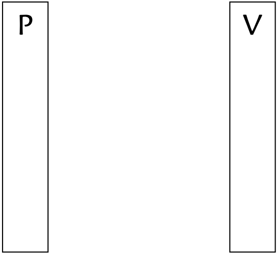
V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.

$x \xleftarrow{\$} \mathbb{Z}_q$



Scenario

P sends $A = g^k$ where
$k \xleftarrow{\$} \mathbb{Z}_q$
V sends $c \xleftarrow{\$} \mathbb{Z}_q$
P sends $s = k + cx \mod q$
V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.

$x \xleftarrow{\$} \mathbb{Z}_q$

$h = g^x$

P

V

Scenario

P sends $A = g^k$ where

$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

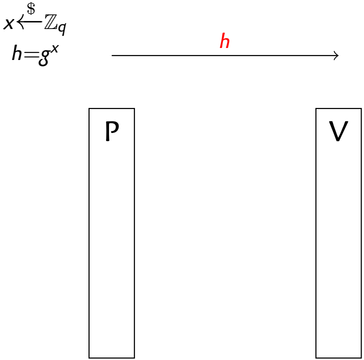
P sends $s = k + cx \mod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where

$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

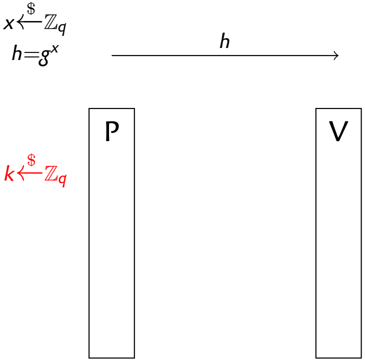
P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where

$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

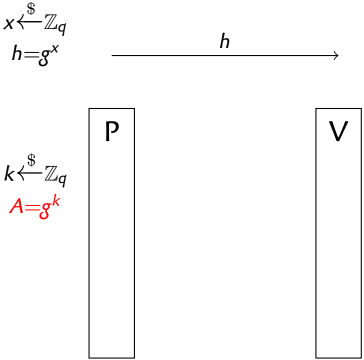
P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of **prime** order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where

$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

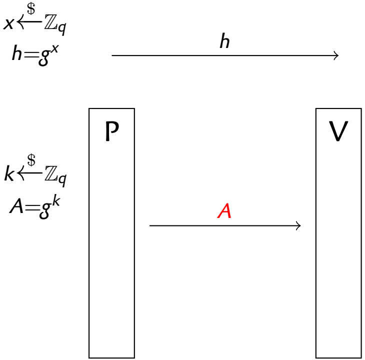
P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where
$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

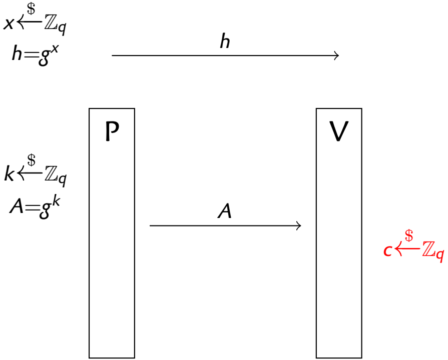
P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where

$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

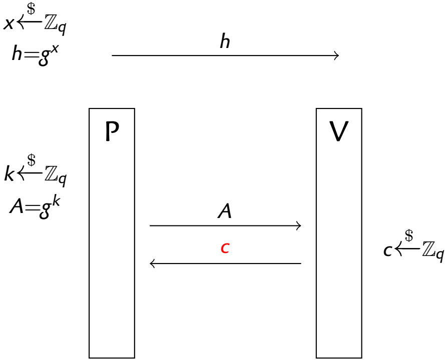
P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of **prime** order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where

$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

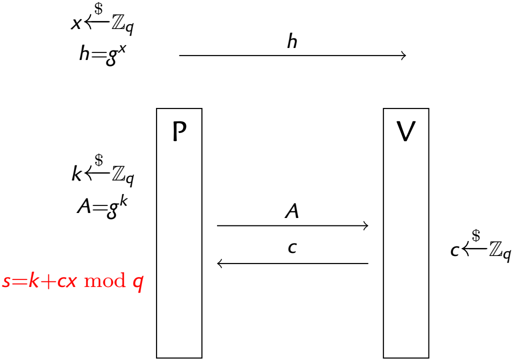
P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of **prime** order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

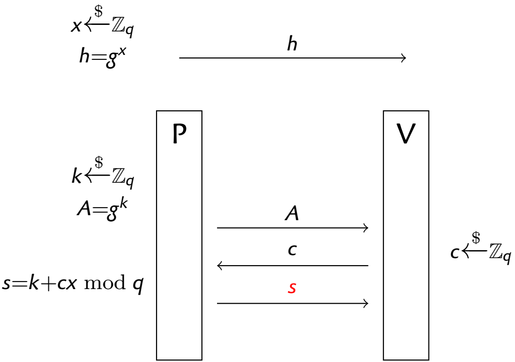
P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

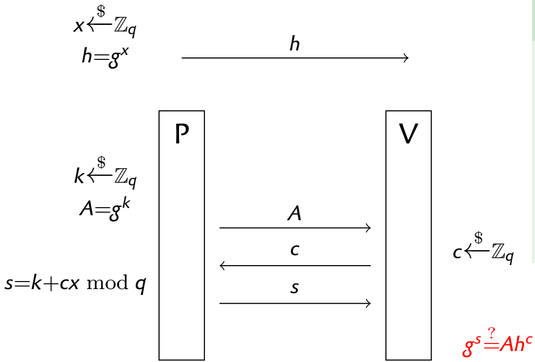
P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol (1991)

Let $\langle g \rangle$ be a group of prime order q

Prover P proves to **verifier** V that she **knows** the **discrete log** x of a public group element $h = g^x$.



Scenario

P sends $A = g^k$ where

$k \xleftarrow{\$} \mathbb{Z}_q$

V sends $c \xleftarrow{\$} \mathbb{Z}_q$

P sends $s = k + cx \bmod q$

V checks whether $g^s = Ah^c$

Schnorr's Identification Protocol: Does the Secret Leak?

P (prover)	Message	V (verifier)
$k \xleftarrow{\$} \mathbb{Z}_q, A \leftarrow g^k$	$\xrightarrow{A}$	
	$\xleftarrow{c}$	$c \xleftarrow{\$} \mathbb{Z}_q$
$s \leftarrow k + cx \bmod q$	$\xrightarrow{s}$	$g^s = Ah^c$

When the protocol succeeds...

- A passive adversary sees a **transcript** (A, c, s), where
  - A is uniformly random in $\langle g \rangle$
  - c is uniformly random in $\mathbb{Z}_q$
  - $g^s = Ah^c$

Producing valid transcripts **does not require** x

- $s \xleftarrow{\$} \mathbb{Z}_q, c \xleftarrow{\$} \mathbb{Z}_q$ and $A \leftarrow g^s h^{-c}$ (r is uniformly random)

Schnorr's Identification Protocol: Security Arguments

The protocol is **zero-knowledge**

- A **simulator** that just receives h can produce valid transcripts that are indistinguishable from the interactions with a real prover (who knows x)

$\Longrightarrow$ Passive adversaries learn **nothing** about x

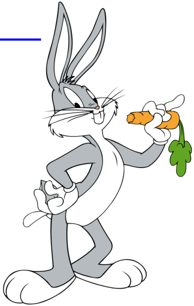
Schnorr's Identification Protocol: Does It Prove Anything?

$h = g^{x}$

Honest Prover

Verifier

x



$$A := g^{k} \longrightarrow$$

$$\longleftarrow c$$

$$s := k + cx \longrightarrow$$

$g^{s} \stackrel{?}{=} Ah^{c}$

YES

Schnorr's Identification Protocol: Does It Prove Anything?

$h = g^{x}$

Malicious Prover

Verifier

A →

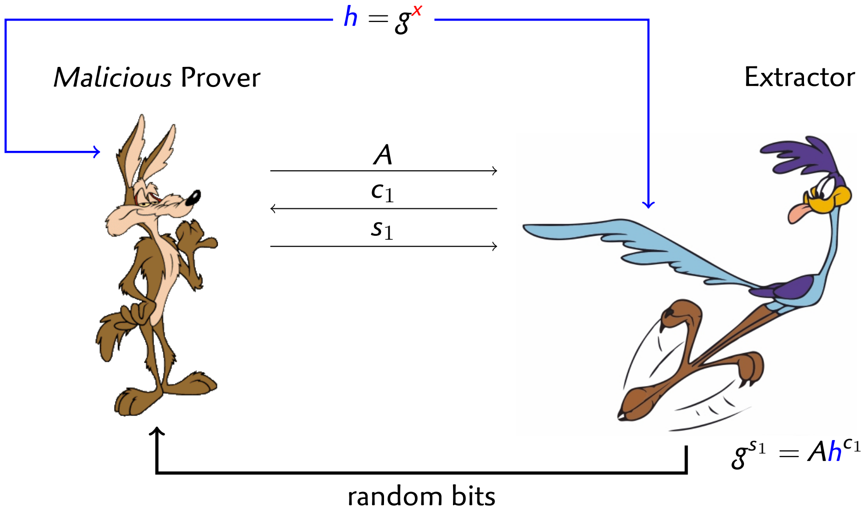
c ←

s →

$g^s \stackrel{?}{=} Ah^c$

YES

Schnorr's Identification Protocol: Does It Prove Anything?



Schnorr's Identification Protocol: Does It Prove Anything?

$h = g^{x}$

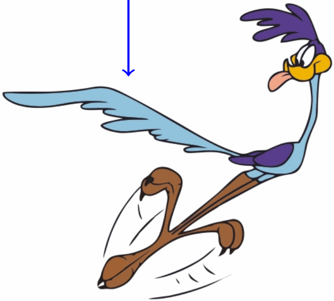
Malicious Prover

Extractor

$$\xrightarrow{\quad A \quad}$$

$$\xleftarrow{\quad c_1 \quad}$$

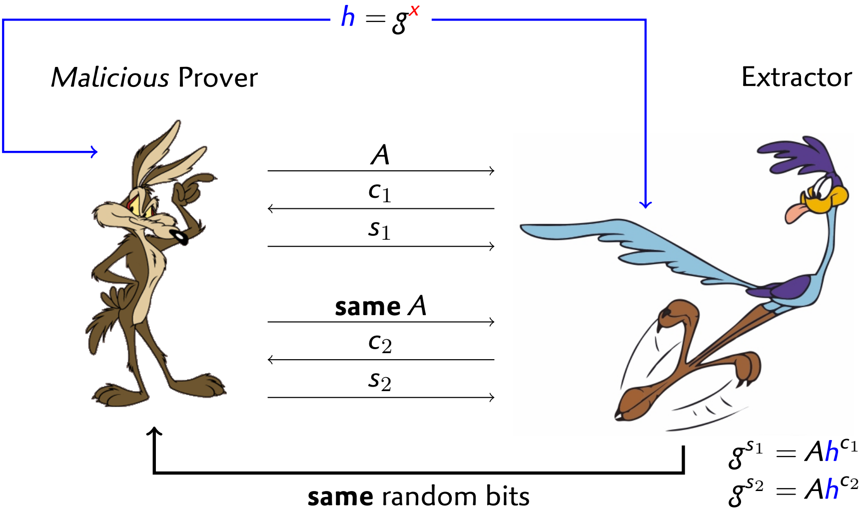
$$\xrightarrow{\quad s_1 \quad}$$



RESET

$$g^{s_1} = A h^{c_1}$$

Schnorr's Identification Protocol: Does It Prove Anything?



Schnorr's Identification Protocol: Does It Prove Anything?

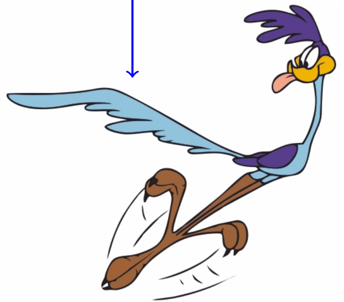
$h = g^{x}$

Malicious Prover

Extractor

$$\xrightarrow{A}$$
$$\xleftarrow{c_1}$$
$$\xrightarrow{s_1}$$

$$\xrightarrow{\textbf{same } A}$$
$$\xleftarrow{c_2}$$
$$\xrightarrow{s_2}$$



$$g^{s_1} = Ah^{c_1}$$
$$g^{s_2} = Ah^{c_2}$$

Combining the two...

$$h^{c_1 - c_2} = g^{s_1 - s_2} \quad \rightsquigarrow \quad h = g^{\frac{s_1 - s_2}{c_1 - c_2} \bmod q}$$

Schnorr's Identification Protocol: Does It Prove Anything?

$h = g^x$

Malicious Prover

A

c_1

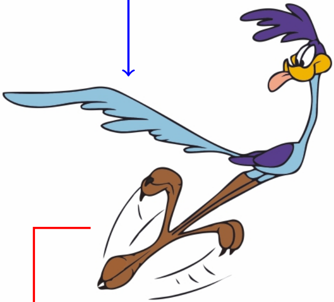
s_1

A

c_2

s_2

Extractor



$$g^{s_1} = Ah^{c_1}$$
$$g^{s_2} = Ah^{c_2}$$

Combining the two...

$$h^{c_1 - c_2} = g^{s_1 - s_2} \quad \rightsquigarrow \quad h = g^{\frac{s_1 - s_2}{c_1 - c_2} \bmod q}$$

$$x = \frac{s_1 - s_2}{c_1 - c_2} \bmod q$$

Schnorr's Identification Protocol: Security Arguments

The protocol is **zero-knowledge**

- A **simulator** that just receives h can produce valid transcripts that are indistinguishable from the interactions with a real prover (who knows x)

$\Longrightarrow$ Passive adversaries learn **nothing** about x

The protocol is **correct**

- Suppose a **malicious prover** (that just receives h) is always accepted by the verifier
- An **extractor** can use it to retrieve x with low overhead

$\Longrightarrow$ **DLOG** is hard $\Longleftrightarrow$ successful prover "knows" x

Fiat, Shamir (1986)

How to Prove Yourself: Practical Solutions to Identification and Signature Problems.
Advances in Cryptology - Crypto'86, Lect. Notes Comput. Science 263, pp. 186-194.

- In such a 3-pass identification scheme, the messages are called **commitment**, **challenge** and **response**. The challenge is randomly chosen by V.

Fiat-Shamir Transform

Replace the challenge by a hash value taken on scheme parameters and the commitment, thereby removing V.
This transforms the protocol by making it **non-interactive**.

The intuition is that any "sufficiently random" hash function should preserve the security of the protocol.

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.

P

Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

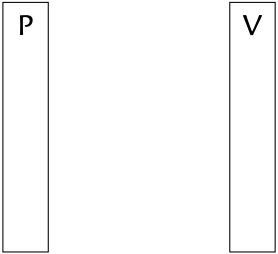
Ver

V checks if $\mathcal{H}(m, g^s \cdot y^{-c}) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

Ver

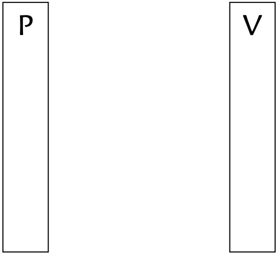
V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.

$x \xleftarrow{\$} \mathbb{Z}_q$



Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

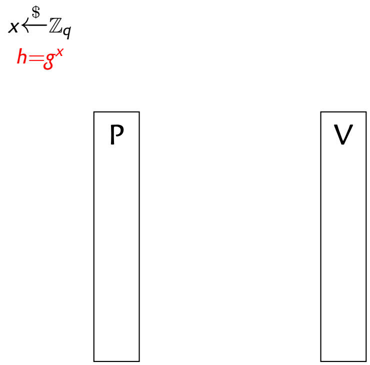
Ver

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \mod q$
P sends $\sigma = (s, c)$

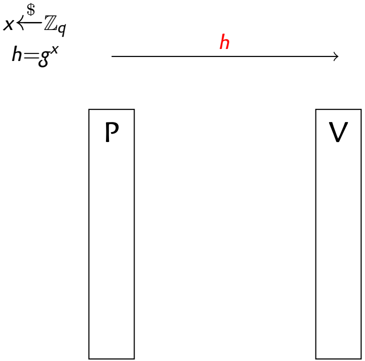
Ver

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \mod q$
P sends $\sigma = (s, c)$

Ver

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.

$x \xleftarrow{\$} \mathbb{Z}_q$

$h = g^x \xrightarrow{\quad h \quad}$

P | V

$k \xleftarrow{\$} \mathbb{Z}_q$

Signing and Verifying

SIGN

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \mod q$
P sends $\sigma = (s, c)$

VER

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.

$x \xleftarrow{\$} \mathbb{Z}_q$

$h = g^x \xrightarrow{\quad h \quad}$

P V

$k \xleftarrow{\$} \mathbb{Z}_q$

$A = g^k$

Signing and Verifying

SIGN

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

P computes $c = \mathcal{H}(m, A)$

P computes $s = k + cx \mod q$

P sends $\sigma = (s, c)$

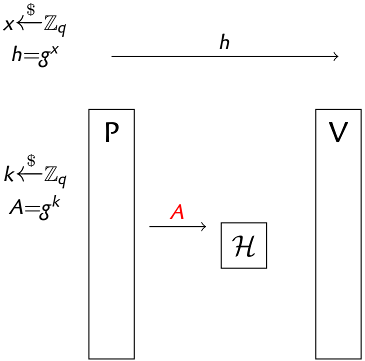
VER

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \mod q$
P sends $\sigma = (s, c)$

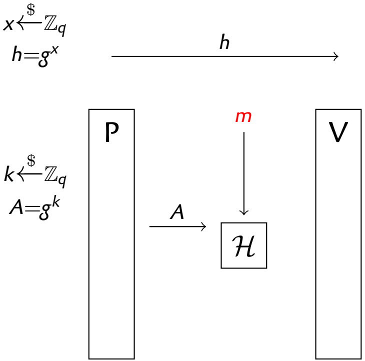
Ver

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

P computes $c = \mathcal{H}(m, A)$

P computes $s = k + cx \mod q$

P sends $\sigma = (s, c)$

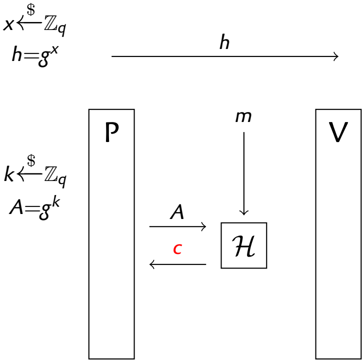
Ver

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_\mathcal{H}$ is a tuple of probabilistic algorithms $\Sigma_\mathcal{H} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$

P computes $c = \mathcal{H}(m, A)$

P computes $s = k + cx \bmod q$

P sends $\sigma = (s, c)$

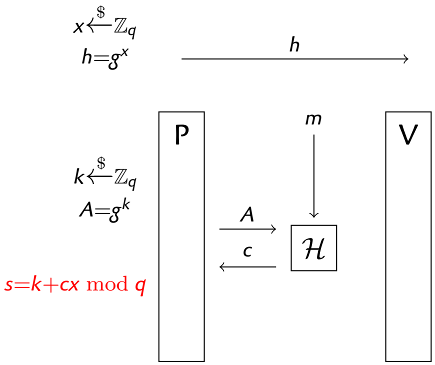
$\textsc{Ver}$

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

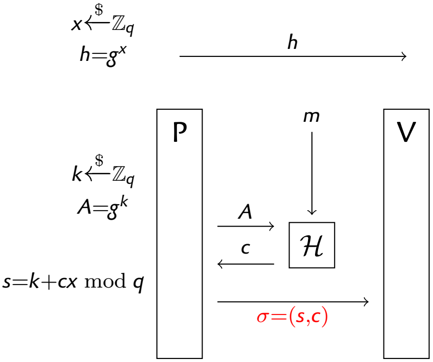
$\textsc{Ver}$

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

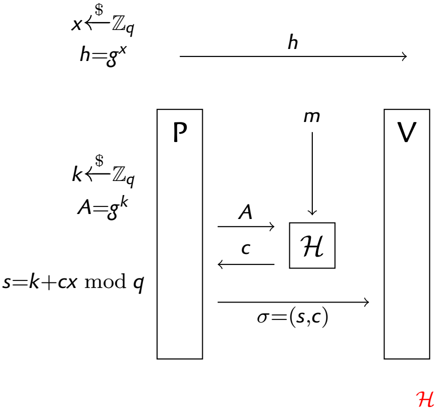
$\textsc{Ver}$

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_\mathcal{H}$ is a tuple of probabilistic algorithms $\Sigma_\mathcal{H} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

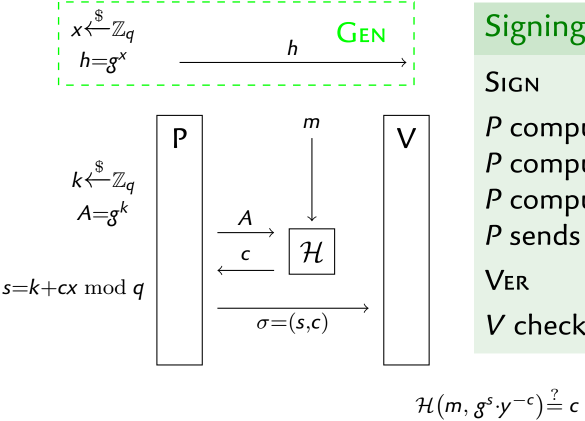
$\textsc{Ver}$

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

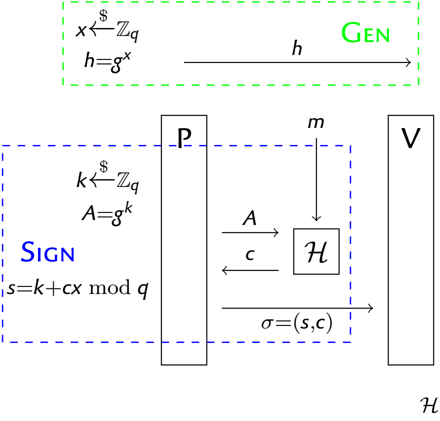
Ver

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

Sign

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \mod q$
P sends $\sigma = (s, c)$

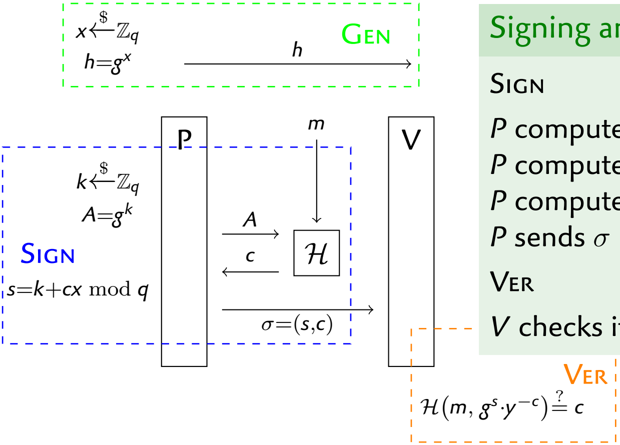
Ver

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Schnorr Signatures (via the Fiat-Shamir Transform)

Introduce a hash function $\mathcal{H} : \{0,1\}^\star \mapsto \mathbb{Z}_q$

Schnorr's signature scheme $\Sigma_{\mathcal{H}}$ is a tuple of probabilistic algorithms $\Sigma_{\mathcal{H}} = (\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ defined as follows.



Signing and Verifying

$\textsc{Sign}$

P computes $A = g^k$ where $k \xleftarrow{\$} \mathbb{Z}_q$
P computes $c = \mathcal{H}(m, A)$
P computes $s = k + cx \bmod q$
P sends $\sigma = (s, c)$

$\textsc{Ver}$

V checks if $\mathcal{H}\left(m, g^s \cdot y^{-c}\right) = c$

Claus Peter Schnorr
(1943–)

More Schnorr-Style Proofs

Base Interactive Protocol

- Verifier knows g and h
- Prover demonstrates **knowledge** of x such that $h = g^x$

Non-Interactive Version via the Fiat-Shamir Transform

- Prover generates a **proof** (a bit string) that the verifier checks

Applications

- Proving knowledge of a secret key (identification protocol)
- Proving validity of DH quadruplet
- Elgamal encryption:
  - Proving knowledge of the randomness (g^r, mh^r)
  - Proving correct decryption
  - Proving encryption of 0/1

Reminder: Decisional Diffie-Hellman

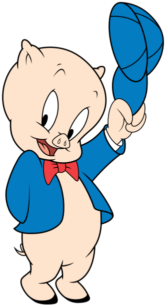
(g, h, g^x, h^y)

(g, h, g^x, h^x)

Proof of DH Quadruplet: Chaum-Pedersen Protocol (1992)

$$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, u, v)$$

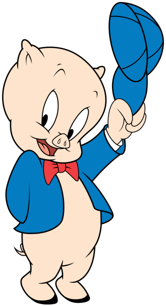
x



Proof of DH Quadruplet: Chaum-Pedersen Protocol (1992)

$(g, h, g^x, h^x) \dashrightarrow (g, h, u, v)$

x



$\xrightarrow{A := g^k,\, B := h^k}$

$\xleftarrow{c}$

$\xrightarrow{s := k + cx}$

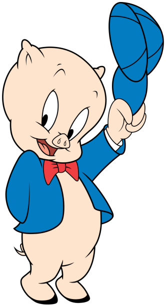
$g^s \stackrel{?}{=} Au^c$

$h^s \stackrel{?}{=} Bv^c$

Proof of DH Quadruplet: Chaum-Pedersen Protocol (1992)

$$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, g^{x}, v)$$

x



$$A := g^k, B := h^k \longrightarrow$$

$$c \longleftarrow$$

$$s := k + cx \longrightarrow$$

$$g^s = Au^c$$

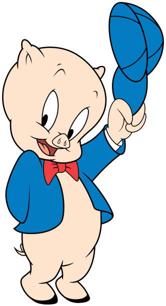
$$h^s \stackrel{?}{=} Bv^c$$

Proof of DH Quadruplet: Chaum-Pedersen Protocol (1992)

(g, h, g^{x}, h^{x}) $\dashrightarrow$ (g, h, g^{x}, h^{x})

$\uparrow$ $\downarrow$

x



$\xrightarrow{A := g^k,\, B := h^k}$

$\xleftarrow{c}$

$\xrightarrow{s := k + cx}$

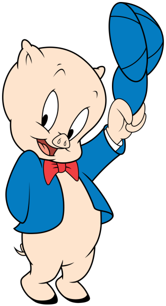
$g^s = Au^c$

$h^s = Bv^c$

Proof of DH Quadruplet: Chaum-Pedersen Protocol (1992)

$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, g^{x}, h^{x})$

x



$$\xrightarrow{A := g^k,\, B := h^k}$$

$$\xleftarrow{c}$$

$$\xrightarrow{s := k + cx}$$

$g^s = Au^c$

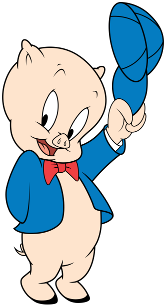
$h^s = Bv^c$

Proof of DH Quadruplet: Chaum-Pedersen Protocol (1992)

$$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, g^{x}, h^{x})$$

$\uparrow$ $\qquad\qquad\qquad\qquad$ $\downarrow$

x



$$\xrightarrow{A := g^k,\ B := h^k}$$

$$\xleftarrow{c}$$

$$\xrightarrow{s := k + cx}$$

$$g^s = Au^c$$

$$h^s = Bv^c$$

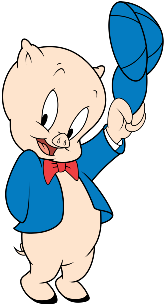
Application: proof of correct Elgamal decryption

- Is m the Elgamal decryption of (α, β)? $\qquad h = g^{x}, (g^r, mh^r)$
- Run above protocol on $(g, \alpha, h, \beta/m)$ with witness x

Non-Interactive Proof of DH Quadruplet

$$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, u, v)$$

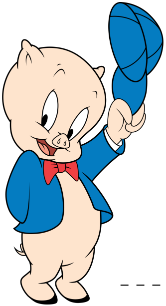
x



Non-Interactive Proof of DH Quadruplet

$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, u, v)$

x



$k \xleftarrow{\$} \mathbb{Z}_q$

$A \leftarrow g^k$

$B \leftarrow h^k$

$c \leftarrow \mathcal{H}(g, h, g^{x}, h^{x}, A, B)$

$s \leftarrow k + cx$

$(c, s) \dashrightarrow$

$g^s \stackrel{?}{=} Au^c$

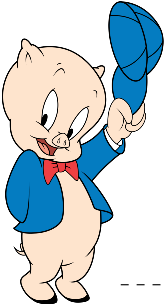
$h^s \stackrel{?}{=} Bv^c$

$\mathcal{H}(g, h, u, v, g^s u^{-c}, h^s v^{-c}) \stackrel{?}{=} c$

Non-Interactive Proof of DH Quadruplet

$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, u, v)$

x



$k \xleftarrow{\$} \mathbb{Z}_q$

$A \leftarrow g^k$

$B \leftarrow h^k$

$c \leftarrow \mathcal{H}(g, h, g^{x}, h^{x}, A, B)$

$s \leftarrow k + cx$

$\xrightarrow{(c, s)}$

$g^s \stackrel{?}{=} Au^c$

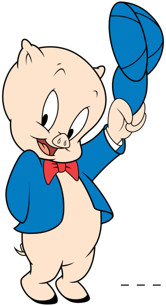
$h^s \stackrel{?}{=} Bv^c$

$\mathcal{H}(g, h, u, v, g^s u^{-c}, h^s v^{-c}) = c$

Non-Interactive Proof of DH Quadruplet

$$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, u, v)$$

x



$$k \xleftarrow{\$} \mathbb{Z}_q$$

$$A \leftarrow g^k$$

$$B \leftarrow h^k$$

$$c \leftarrow \mathcal{H}(g, h, g^{x}, h^{x}, A, B)$$

$$s \leftarrow k + cx$$

$$\xrightarrow{(c, s)}$$

$$g^s = Au^c$$

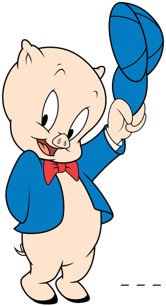
$$h^s = Bv^c$$

$$\mathcal{H}(g, h, u, v, g^s u^{-c}, h^s v^{-c}) = c$$

Non-Interactive Proof of DH Quadruplet

$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, g^{x}, h^{x})$

x



$k \xleftarrow{\$} \mathbb{Z}_q$

$A \leftarrow g^k$

$B \leftarrow h^k$

$c \leftarrow \mathcal{H}(g, h, g^{x}, h^{x}, A, B)$

$s \leftarrow k + cx$

(c, s)

$g^s = Au^c$

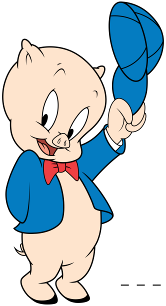
$h^s = Bv^c$

$\mathcal{H}(g, h, u, v, g^s u^{-c}, h^s v^{-c}) = c$

Non-Interactive Proof of DH Quadruplet

$(g, h, g^{x}, h^{x}) \dashrightarrow (g, h, g^{x}, h^{x})$

x



$k \xleftarrow{\$} \mathbb{Z}_q$

$A \leftarrow g^k$

$B \leftarrow h^k$

$c \leftarrow \mathcal{H}(g, h, g^{x}, h^{x}, A, B)$

$s \leftarrow k + cx$

(c, s)

$\mathcal{H}(g, h, u, v, g^s u^{-c}, h^s v^{-c}) = c$

Concrete Problem: e-Voting with Strong Security Guarantees

- **Voters** register in advance at the **polling station**
  - Polling station know the public key $h_v = g^{x_v}$ of voter v
- The **election administrator** publishes a public key $h_{ea} = g^{x_{ea}}$
- The **polling station** publishes a public key $h_{ps} = g^{x_{ps}}$
- Yes/No question: votes are either 0 (no) or 1 (yes)
- On voting day, voter v has a **vote** $b_v \in \{0, 1\}$, and:
  - 👍 **Identify** to the polling station using h_v
  - 👍 Elgamal-**encrypt** the **ballot** g^{b_v} using h_{ea}
  - 👍 **Signs** the encrypted ballot using their secret key x_v
  - Sends their encrypted/signed ballot to the polling station
  - 👍 The polling station **signs** incoming ballots using x_{ps}, send back the signed ballot to voters and publishes everything
- At the end of the day:
  - The polling station compute the **product** π of all votes
  - Publishes π and send π to the election administrator
  - Election administrator Elgamal-**decrypts** π
  - Election administrator publishes the number of "1" votes

Elgamal encryption is **malleable**

- Product of encryption is encryption of product
  - If $(\alpha, \beta) = (g^r, m_1 h^r)$ and $(\gamma, \delta) = (g^t, m_2 h^t)$
  - Then $(\alpha\gamma, \beta\delta) = (g^{r+t}, m_1 m_2 h^{r+t})$
- product of encryptions of g^{b_v} is encryption of $g^{\sum_v b_v}$

Security Guarantees

- Only registered voters may send a ballot
  - Voter v must prove knowledge of x_v
- Polling station does not know the votes
  - Semantic security of Elgamal encryption
- Polling station cannot modify a ballot
  - They are signed by the voters
- Polling station cannot “forget” a ballot
  - Voters may exhibit their ballot signed by the polling station
- Correct value of π is publicly verifiable

Problems

1. Votes are not private from the election authority
   - Election authority knows the decryption key x_{ea}...
2. The election authority could cheat
   - In theory, decrypts π using x_{ea}, publishes result
   - π is an encryption of $g^{\sum_v b_v}$
   - What if the election authority publishes a different value?
   - 👍 (non-interactive) proof of correct Elgamal decryption
3. Voters could cheat by submitting incorrect ballots
   - Instead of encrypting g^0 or g^1, a voter encrypts g^{1000}
   - $\rightsquigarrow$ Votes $1000\times$ "yes"!!!
   - Detectable: wrong # votes compared to # ballots
   - But it still voids the election...
4. Voters have a **receipt**
   - They have their ballot signed by the polling station
   - They have a "proof" that they have voted "yes" or "no"
   - They could sell their vote / be jailed / etc.

Guaranteeing Votes Privacy

Main idea

- $n > 1$ election authorities
  - Votes remain confidential as long as **one** of them is honest

Threshold Elgamal

- **Distributed key generation**:
  - Authority $\#i$ samples $x_i \xleftarrow{\$} \mathbb{Z}_q$, sets $h_i \leftarrow g^{x_i}$
  - All authorities publish their **partial public keys** h_i
  - Global public key: $H \leftarrow \prod_i h_i$ ($X = \sum_i x_i$)
- Encryption of m: (g^r, mH^r) as usual (with random r)
- **Distributed decryption** of (α, β):
  - Goal: $m = \beta/\alpha^X$ (X shared between them)
  - Authority #i computes $\gamma_i \leftarrow \alpha^{x_i}$, publishes γ_i
  - The world computes $m \leftarrow \beta / \prod_i \gamma_i$

Guaranteeing Correct Decryption of the Tally

Threshold Elgamal

- **Distributed key generation**:
  - Authority $\#i$ samples $x_i \xleftarrow{\$} \mathbb{Z}_q$, sets $h_i \leftarrow g^{x_i}$
  - All authorities publish their **partial public keys** h_i
  - Global public key: $H \leftarrow \prod_i h_i$ ($X = \sum_i x_i$)
- Encryption of m: (g^r, mH^r) as usual (with random r)
- **Distributed decryption** of (α, β):
  - Goal: $m = \beta / \alpha^X$ (X shared between them)
  - Authority #i computes $\gamma_i \leftarrow \alpha^{x_i}$, publishes γ_i
  - The world computes $m \leftarrow \beta / \prod_i \gamma_i$

Preventing a rogue election authority from cheating

- $(g, \alpha, h_i, \gamma_i)$ is (supposed to be) a DH-quadruplet (with x_i)
- Election authorities publishes (non-interactive) proofs

Proof of 0/1 Encryption: Cramer, Damgård, and Schoenmakers (1994)

Two possible messages: m_0 ="NO" and m_1 ="YES"

- prove that (α, β) is an Elgamal encryption of a valid message
  - (without revealing if it is m_0 or m_1)

$\rightsquigarrow$ prove knowledge of r, b such that $\alpha = g^r$ and $\beta / m_b = h^r$

Strategy

- Prove knowledge of r_0 such that $\alpha = g^{r_0}$ and $\beta / m_0 = h^{r_0}$
- Prove knowledge of r_1 such that $\alpha = g^{r_1}$ and $\beta / m_1 = h^{r_1}$
- Verifier checks **both** proofs, accept if **both** correct 🤨

Obstacle

- Prover can only produce a **valid proof** of knowledge of r_b
  - Simply does not know r_{1-b}... 😵‍💫

Proof of 0/1 Encryption: Cramer, Damgård, and Schoenmakers (1994)

Solution

- Prover generates a **fake proof of knowledge** of r_{1-b} 🤯
  - Without knowing r_{1-b}

Producing fake proofs?!?

- **Correct** protocol $\rightsquigarrow$ no way to generate fake proofs 🤔
  - Verifier rejects fake proofs if DLOG is hard

The basic protocol is **zero-knowledge**

- A **simulator** generates valid transcripts (=proofs) by **choosing the challenge** before committing 😈
  - So we could generate a fake proof of knowledge of r_{1-b} if we could choose the challenge

Proof of 0/1 Encryption: Cramer, Damgård, and Schoenmakers (1994)

Strategy (cont'd)

Prover wants to:

- Use a challenge imposed by the verifier for r_b (honest proof)
- Choose the challenge for r_{1-b} (fake proof)

How to choose the challenge for r_{1-b} but not for r_b? 😵

Prover **splits** the challenge (Schnorr — 1991)

- **Chooses** $c_{1-b} \xleftarrow{\$} \mathbb{Z}_q$ (for the fake proof)
  - ⇝ Produces fake proof **in advance** 🤣
- **commits**
- Receives **unpredictable** challenge c from the verifier
- Computes $c_b \leftarrow c - c_{1-b}$ (for the honest proof)
  - ⇝ produces the honest proof "online" $(c = c_0 + c_1)$
- Sends both proofs plus (c_0, c_1) to the verifier

Prover$(h, b, (\alpha, \beta), r_b)$

- $\boxed{c_{1-b} \xleftarrow{\$} \mathbb{Z}_q}, s_{1-b} \xleftarrow{\$} \mathbb{Z}_q$ — Choose challenge
- $A_{1-b} \leftarrow g^{s_{1-b}} \alpha^{-c_{1-b}}$
- $B_{1-b} \leftarrow h^{s_{1-b}} (\beta / m_{1-b})^{-c_{1-b}}$
- $k \xleftarrow{\$} \mathbb{Z}_q, A_b \leftarrow \alpha^k, B_b \leftarrow h^k$

1. Send A_0, B_0, A_1, B_1

- $c_b \leftarrow c - c_{1-b}, s_b \leftarrow k + c_b r_b$

3. Send c_0, s_0, s_1

Verifier$(h, (\alpha, \beta))$

2. Send $c \xleftarrow{\$} \mathbb{Z}_q$

- $c_1 \leftarrow c - c_0$
- Check $g^{s_0} \stackrel{?}{=} A_0 \alpha^{c_0}$
- Check $h^{s_0} \stackrel{?}{=} B_0 (\beta / m_0)^{c_0}$
- Check $g^{s_1} \stackrel{?}{=} A_1 \alpha^{c_1}$
- Check $h^{s_1} \stackrel{?}{=} B_1 (\beta / m_1)^{c_1}$

Prover$(h, b, (\alpha, \beta), r_b)$

- $c_{1-b} \xleftarrow{\$} \mathbb{Z}_q$, $s_{1-b} \xleftarrow{\$} \mathbb{Z}_q$
- $A_{1-b} \leftarrow g^{s_{1-b}} \alpha^{-c_{1-b}}$
- $B_{1-b} \leftarrow h^{s_{1-b}} (\beta / m_{1-b})^{-c_{1-b}}$
- $k \xleftarrow{\$} \mathbb{Z}_q, A_b \leftarrow \alpha^k, B_b \leftarrow h^k$

1. Send A_0, B_0, A_1, B_1

- $c_b \leftarrow c - c_{1-b}, s_b \leftarrow k + c_b r_b$

3. Send c_0, s_0, s_1

Verifier$(h, (\alpha, \beta))$

Fake proof for

$$\begin{cases} \alpha = g^{r_{1-b}} \\ \beta / m_{1-b} = h^{r_{1-b}} \end{cases}$$

w/ (chosen) challenge c_{1-b}

2. Send $c \xleftarrow{\$} \mathbb{Z}_q$

- $c_1 \leftarrow c - c_0$
- Check $g^{s_0} \stackrel{?}{=} A_0 \alpha^{c_0}$
- Check $h^{s_0} \stackrel{?}{=} B_0 (\beta / m_0)^{c_0}$
- Check $g^{s_1} \stackrel{?}{=} A_1 \alpha^{c_1}$
- Check $h^{s_1} \stackrel{?}{=} B_1 (\beta / m_1)^{c_1}$

Prover$(h, b, (\alpha, \beta), r_b)$

- $c_{1-b} \xleftarrow{\$} \mathbb{Z}_q, s_{1-b} \xleftarrow{\$} \mathbb{Z}_q$
- $A_{1-b} \leftarrow g^{s_{1-b}} \alpha^{-c_{1-b}}$
- $B_{1-b} \leftarrow h^{s_{1-b}} (\beta / m_{1-b})^{-c_{1-b}}$
- $\boxed{k \xleftarrow{\$} \mathbb{Z}_q, A_b \leftarrow \alpha^k, B_b \leftarrow h^k}$

1. Send A_0, B_0, A_1, B_1

- $c_b \leftarrow c - c_{1-b}, \boxed{s_b \leftarrow k + c_b r_b}$

3. Send c_0, s_0, s_1

Verifier$(h, (\alpha, \beta))$

Honest proof for

$$\begin{cases} \alpha = g^{r_b} \\ \beta / m_b = h^{r_b} \end{cases}$$

with challenge c_b

2. Send $c \xleftarrow{\$} \mathbb{Z}_q$

- $c_1 \leftarrow c - c_0$
- Check $g^{s_0} \stackrel{?}{=} A_0 \alpha^{c_0}$
- Check $h^{s_0} \stackrel{?}{=} B_0 (\beta / m_0)^{c_0}$
- Check $g^{s_1} \stackrel{?}{=} A_1 \alpha^{c_1}$
- Check $h^{s_1} \stackrel{?}{=} B_1 (\beta / m_1)^{c_1}$

PROVER$(h, b, (\alpha, \beta), r_b)$

- $c_{1-b} \xleftarrow{\$} \mathbb{Z}_q, s_{1-b} \xleftarrow{\$} \mathbb{Z}_q$
- $A_{1-b} \leftarrow g^{s_{1-b}} \alpha^{-c_{1-b}}$
- $B_{1-b} \leftarrow h^{s_{1-b}} (\beta / m_{1-b})^{-c_{1-b}}$
- $k \xleftarrow{\$} \mathbb{Z}_q, A_b \leftarrow \alpha^k, B_b \leftarrow h^k$

1. $\boxed{\text{Send } A_0, B_0, A_1, B_1}$

- $c_b \leftarrow c - c_{1-b}, s_b \leftarrow k + c_b r_b$

3. Send c_0, s_0, s_1

VERIFIER$(h, (\alpha, \beta))$

Commit to k (and c_{1-b})

2. Send $c \xleftarrow{\$} \mathbb{Z}_q$

- $c_1 \leftarrow c - c_0$
- Check $g^{s_0} \stackrel{?}{=} A_0 \alpha^{c_0}$
- Check $h^{s_0} \stackrel{?}{=} B_0 (\beta / m_0)^{c_0}$
- Check $g^{s_1} \stackrel{?}{=} A_1 \alpha^{c_1}$
- Check $h^{s_1} \stackrel{?}{=} B_1 (\beta / m_1)^{c_1}$

Prover$(h, b, (\alpha, \beta), r_b)$

- $c_{1-b} \xleftarrow{\$} \mathbb{Z}_q, s_{1-b} \xleftarrow{\$} \mathbb{Z}_q$
- $A_{1-b} \leftarrow g^{s_{1-b}} \alpha^{-c_{1-b}}$
- $B_{1-b} \leftarrow h^{s_{1-b}} (\beta / m_{1-b})^{-c_{1-b}}$
- $k \xleftarrow{\$} \mathbb{Z}_q, A_b \leftarrow \alpha^k, B_b \leftarrow h^k$

1. Send A_0, B_0, A_1, B_1

- $\boxed{c_b \leftarrow c - c_{1-b},} s_b \leftarrow k + c_b r_b$

3. Send c_0, s_0, s_1

Verifier$(h, (\alpha, \beta))$

Once c_{1-b} has been committed, c_b cannot be predicted by the prover (proof $\Rightarrow$ knowledge of r_b)

2. Send $c \xleftarrow{\$} \mathbb{Z}_q$

- $c_1 \leftarrow c - c_0$
- Check $g^{s_0} \stackrel{?}{=} A_0 \alpha^{c_0}$
- Check $h^{s_0} \stackrel{?}{=} B_0 (\beta / m_0)^{c_0}$
- Check $g^{s_1} \stackrel{?}{=} A_1 \alpha^{c_1}$
- Check $h^{s_1} \stackrel{?}{=} B_1 (\beta / m_1)^{c_1}$

Elgamal 0/1 Encryption with Non-Interactive Proof

ENCRYPT(h, b)

- $r_b, c_{1-b}, s_{1-b}, k \xleftarrow{\$} \mathbb{Z}_q$
- $(\alpha, \beta) \leftarrow (g^{r_b}, m_b h^{r_b})$
- $(A_{1-b}, B_{1-b}) \leftarrow (g^{s_{1-b}} \alpha^{-c_{1-b}}, h^{s_{1-b}} (\beta / m_{1-b})^{-c_{1-b}})$
- $(A_b, B_b) \leftarrow (\alpha^k, h^k)$
- $c \leftarrow \mathcal{H}(h, \alpha, \beta, A_0, B_0, A_1, B_1)$
- $c_b \leftarrow c - c_{1-b}$
- $s_b \leftarrow k + c_b r_b$
- **return**$(\alpha, \beta, c_0, c_1, s_0, s_1)$

DECRYPT$(x, \alpha, \beta, c_0, c_1, s_0, s_1)$

- $c \leftarrow \mathcal{H}(g^x, \alpha, \beta, g^{s_0} \alpha^{-c_0}, h^{s_0} (m_0/\beta)^{c_0}, g^{s_1} \alpha^{-c_1}, h^{s_1} (m_1/\beta)^{c_1})$
- **if** $c = c_0 + c_1 \pmod{q}$ **then return** β / α^x **else return** $\perp$

Digital Signature Algorithm (DSA)

- The Digital Signature Algorithm (DSA) is a United States Federal Government standard or FIPS for digital signatures.
- It was proposed by the National Institute of Standards and Technology (NIST) in **August 1991** for use in their Digital Signature Standard (DSS), specified in FIPS 186, adopted in **1993**.
- DSA makes use of a cryptographic hash function $\mathcal{H}$.
- 2025: ECDSA with $\mathcal{H}$:= SHA256 is widespread

Textbook ElGamal signature scheme (1985)

Public parameters. A k-bit prime p and a generator g of $\mathbb{Z}_p^\times$

Key generation. The secret key is $x \xleftarrow{\$} \mathbb{Z}_{p-1}$
The public key is $y = g^x \bmod p$

Signature. To sign a message $m \in \mathbb{Z}_{p-1}$, generate (r, s) s.t.

$$g^m = y^r r^s \bmod p$$

as follows: $k \xleftarrow{\$} \mathbb{Z}_{p-1}^\times$, $r \leftarrow g^k \mod p$ and

$$s \leftarrow (m - xr) \cdot k^{-1} \mod p - 1$$

Output (r, s)

Verification. Verify that $1 < r < p$ and $g^m \stackrel{?}{=} y^r r^s \mod p$

Digital Signature Algorithm (DSA)

Hashed ElGamal signature scheme

Public parameters. A k-bit prime p and a generator g of $\mathbb{Z}_p^\times$

Key generation. The secret key is $x \xleftarrow{\$} \mathbb{Z}_{p-1}$
The public key is $y = g^x \mod p$

Signature. To sign a message $m \in \{0,1\}^*$, generate (r, s) s.t.

$$g^{\mathcal{H}(m)} = y^r r^s \mod p$$

as follows: $k \xleftarrow{\$} \mathbb{Z}_{p-1}^\times$, $r \leftarrow g^k \mod p$ and

$$ks \leftarrow (\mathcal{H}(m) - xr) \cdot k^{-1} \mod p-1$$

Output (r, s)

Verification. Verify that $1 < r < p$ and $g^{\mathcal{H}(m)} \stackrel{?}{=} y^r r^s \mod p$

Digital Signature Algorithm (DSA)

Hashed ElGamal signature scheme with Schnorr's trick

Public parameters. A k-bit prime p and a generator $g \in \mathbb{Z}_p^\times$ of prime order q

Key generation. The secret key is $x \xleftarrow{\$} \mathbb{Z}_q$
The public key is $y = g^x \mod p$

Signature. To sign a message $m \in \{0,1\}^*$, generate (r, s) s.t.

$$g^{\mathcal{H}(m)} = y^r r^s \mod p$$

as follows: $k \xleftarrow{\$} \mathbb{Z}_q^\times$, $r \leftarrow g^k \mod p$ and

$$s \leftarrow (\mathcal{H}(m) - xr) \cdot k^{-1} \mod q$$

Output (r, s)

Verification. Verify that $1 < r < q$ and $g^{\mathcal{H}(m)} \stackrel{?}{=} y^r r^s \mod p$

Digital Signature Algorithm (DSA)

Full DSA

Public parameters. A k-bit prime p and a generator $g \in \mathbb{Z}_p^\times$ of prime order q

Key generation. The secret key is $x \xleftarrow{\$} \mathbb{Z}_q$
The public key is $y = g^x \mod p$

Signature. To sign a message $m \in \mathbb{Z}_{p-1}$, generate (r, s) s.t.

$$g^{\mathcal{H}(m)} = y^r r^s \mod p$$

as follows: $k \xleftarrow{\$} \mathbb{Z}_q^\times$, $r \leftarrow (g^k \mod p) \mod q$ and

$$s \leftarrow (\mathcal{H}(m) + xr) \cdot k^{-1} \mod q$$

Output (r, s)

Verification. Verify that $1 < r < q$, compute $w \leftarrow s^{-1} \mod q$,
$u_1 = \mathcal{H}(m) \cdot w \mod q$, $u_2 \leftarrow r \cdot w \mod q$,
Check whether $(g^{u_1} y^{u_2} \mod p) \mod q \stackrel{?}{=} r$