Commitment Schemes and Zero-knowledge proofs

Benoît Libert
benoit.libert@ens-lyon.fr

Commitment schemes

The coin flipping problem:

- Two distrustful parties want to play a coin flipping game over the Internet or by phone
- . . . or even jointly generate a sequence of random bits
- How can they make sure the other party is not cheating?
- **Solution:** Use a cryptographic commitment scheme

Commitments

Digital equivalent of a sealed box

What does it provide?

- **Binding** property: once I have sent a value in a locked box, I cannot change it anymore
- **Hiding** property: nobody can tell what is inside the box without the key

Commitments

- In commitments schemes $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$,
  - $\mathsf{Setup}(\lambda)$ given a security parameter $\lambda \in \mathbb{N}$, outputs a public key pk
  - $\mathsf{Com}_{pk}(m)$ outputs a *commitment com* and a *decommitment dec*
  - $\mathsf{Open}_{pk}(com, dec)$ outputs evidence dec that the committed message was m
- Requirements:
  - **Hiding**: for any $m_0, m_1 \in \mathcal{M}$, we have $\{\mathsf{Com}_{pk}(m_0)\} \approx \{\mathsf{Com}_{pk}(m_1)\}$
  - **Binding**: given pk, it must be infeasible to output com and two correct openings (m, dec), (m', dec') with $m \neq m'$

Commitments

Pedersen's commitment:

- Setup(λ) chooses a group $\mathbb{G}$ of prime order $q > 2^\lambda$ and $g, h \xleftarrow{R} \mathbb{G}$. It defines $pk = (g, h)$
- $\mathrm{Com}_{pk}(m)$ outputs $com = g^m \cdot h^r$, with $r \xleftarrow{R} \mathbb{Z}_q$, and sets $dec = (m, r)$
- $\mathrm{Open}_{pk}(com, dec)$ returns $dec = (m, r)$; verifier accepts if $com = g^m \cdot h^r$
- **Hiding** property is unconditional
- **Binding** property relies on the *discrete logarithm* problem:

  Distinct openings (m, r), (m', r') of a given commitment $com = g^m h^r = g^{m'} h^{r'}$ reveal

$$\log_g(h) = (r' - r)/(m - m') \bmod q$$

Commitments

Pedersen's commitment:

- Setup(λ) chooses a group $\mathbb{G}$ of prime order $q > 2^\lambda$ and $g, h \stackrel{R}{\leftarrow} \mathbb{G}$. It defines $pk = (g, h)$
- $\mathrm{Com}_{pk}(m)$ outputs $com = g^m \cdot h^r$, with $r \stackrel{R}{\leftarrow} \mathbb{Z}_q$, and sets $dec = (m, r)$
- $\mathrm{Open}_{pk}(com, dec)$ returns $dec = (m, r)$; verifier accepts if $com = g^m \cdot h^r$
- **Hiding** property is unconditional
- **Binding** property relies on the *discrete logarithm* problem:

  Distinct openings (m, r), (m', r') of a given commitment $com = g^m h^r = g^{m'} h^{r'}$ reveal

$$\log_g(h) = (r' - r)/(m - m') \bmod q$$

Commitments

Pedersen's commitment:

- Setup(λ) chooses a group $\mathbb{G}$ of prime order $q > 2^\lambda$ and $g, h \stackrel{R}{\leftarrow} \mathbb{G}$. It defines $pk = (g, h)$
- $\mathrm{Com}_{pk}(m)$ outputs $com = g^m \cdot h^r$, with $r \stackrel{R}{\leftarrow} \mathbb{Z}_q$, and sets $dec = (m, r)$
- $\mathrm{Open}_{pk}(com, dec)$ returns $dec = (m, r)$; verifier accepts if $com = g^m \cdot h^r$
- **Hiding** property is unconditional
- **Binding** property relies on the *discrete logarithm* problem:

  Distinct openings (m, r), (m', r') of a given commitment $com = g^m h^r = g^{m'} h^{r'}$ reveal

$$\log_g(h) = (r' - r)/(m - m') \bmod q$$

Commitments

Pedersen's commitment:

- Setup(λ) chooses a group $\mathbb{G}$ of prime order $q > 2^\lambda$ and $g, h \stackrel{R}{\leftarrow} \mathbb{G}$. It defines $pk = (g, h)$
- $\mathsf{Com}_{pk}(m)$ outputs $com = g^m \cdot h^r$, with $r \stackrel{R}{\leftarrow} \mathbb{Z}_q$, and sets $dec = (m, r)$
- $\mathsf{Open}_{pk}(com, dec)$ returns $dec = (m, r)$; verifier accepts if $com = g^m \cdot h^r$
- **Hiding** property is unconditional
- **Binding** property relies on the *discrete logarithm* problem:

  Distinct openings (m, r), (m', r') of a given commitment $com = g^m h^r = g^{m'} h^{r'}$ reveal

$$\log_g(h) = (r' - r)/(m - m') \bmod q$$

Commitments

RSA-based commitment:

- Setup(λ) chooses an RSA modulus $N = pq$, with a prime e s.t. $\gcd(e, \varphi(N)) = 1$ and $g \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$. It defines $pk = (g, e, N)$
- $\mathrm{Com}_{pk}(m)$ given $m \in \{0, \ldots, e-1\}$, outputs $com = g^m \cdot r^e \bmod N$, with $r \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$, and sets $dec = (m, r)$
- $\mathrm{Open}_{pk}(m, d)$ returns $dec = (m, r)$; verifier accepts if $com = g^m \cdot r^e \bmod N$
- Unconditionally **hiding**
- **Binding** under the *RSA assumption*: two distinct openings (m, r), (m', r') such that

$$g^m r^e \equiv g^{m'} r'^e \pmod N$$

reveal $g^{1/e} \bmod N$

Commitments

RSA-based commitment:

- Setup(λ) chooses an RSA modulus $N = pq$, with a prime e s.t. $\gcd(e, \varphi(N)) = 1$ and $g \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$. It defines $pk = (g, e, N)$
- $\mathsf{Com}_{pk}(m)$ given $m \in \{0, \ldots, e-1\}$, outputs $com = g^m \cdot r^e \bmod N$, with $r \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$, and sets $dec = (m, r)$
- $\mathsf{Open}_{pk}(m, d)$ returns $dec = (m, r)$; verifier accepts if $com = g^m \cdot r^e \bmod N$
- Unconditionally **hiding**
- **Binding** under the *RSA assumption*: two distinct openings (m, r), (m', r') such that

$$g^m r^e \equiv g^{m'} r'^e \pmod{N}$$

reveal $g^{1/e} \bmod N$

Commitments

RSA-based commitment:

- Setup(λ) chooses an RSA modulus $N = pq$, with a prime e s.t. $\gcd(e, \varphi(N)) = 1$ and $g \xleftarrow{R} \mathbb{Z}_N^*$. It defines $pk = (g, e, N)$
- $\mathrm{Com}_{pk}(m)$ given $m \in \{0, \ldots, e-1\}$, outputs $com = g^m \cdot r^e \bmod N$, with $r \xleftarrow{R} \mathbb{Z}_N^*$, and sets $dec = (m, r)$
- $\mathrm{Open}_{pk}(m, d)$ returns $dec = (m, r)$; verifier accepts if $com = g^m \cdot r^e \bmod N$
- Unconditionally **hiding**
- **Binding** under the *RSA assumption*: two distinct openings (m, r), (m', r') such that

$$g^m r^e \equiv g^{m'} r'^e \pmod{N}$$

reveal $g^{1/e} \bmod N$

Application: Coin-flipping over the Internet

Distrustful parties A and B want to jointly generate a random $b \in_R \{0,1\}$:

- B picks a random $b_B \in_R \{0,1\}$ which is kept secret
- A chooses $b_A \in_R \{0,1\}$, computes a commitment-decommitment pair $(com, dec) = \mathrm{Com}(b_A)$ and sends com to B
- B reveals b_B
- A and B output $b = b_A \oplus b_B$.

Output b is guaranteed to be uniform in $\{0,1\}$ as long as A *or* B is honest

Application: Coin-flipping over the Internet

Distrustful parties A and B want to jointly generate a random $b \in_R \{0,1\}$:

- B picks a random $b_B \in_R \{0,1\}$ which is kept secret
- A chooses $b_A \in_R \{0,1\}$, computes a commitment-decommitment pair $(com, dec) = \mathrm{Com}(b_A)$ and sends com to B
- B reveals b_B
- A and B output $b = b_A \oplus b_B$.

Output b is guaranteed to be uniform in $\{0,1\}$ as long as A *or* B is honest

Application: Coin-flipping over the Internet

Distrustful parties A and B want to jointly generate a random $b \in_R \{0,1\}$:

- B picks a random $b_B \in_R \{0,1\}$ which is kept secret
- A chooses $b_A \in_R \{0,1\}$, computes a commitment-decommitment pair $(com, dec) = \mathrm{Com}(b_A)$ and sends com to B
- B reveals b_B
- A and B output $b = b_A \oplus b_B$.

Output b is guaranteed to be uniform in $\{0,1\}$ as long as A *or* B is honest

Application: Coin-flipping over the Internet

Distrustful parties A and B want to jointly generate a random $b \in_R \{0,1\}$:

- B picks a random $b_B \in_R \{0,1\}$ which is kept secret
- A chooses $b_A \in_R \{0,1\}$, computes a commitment-decommitment pair $(com, dec) = \mathrm{Com}(b_A)$ and sends com to B
- B reveals b_B
- A and B output $b = b_A \oplus b_B$.

Output b is guaranteed to be uniform in $\{0,1\}$ as long as A *or* B is honest

Application: Coin-flipping over the Internet

Distrustful parties A and B want to jointly generate a random $b \in_R \{0,1\}$:

- B picks a random $b_B \in_R \{0,1\}$ which is kept secret
- A chooses $b_A \in_R \{0,1\}$, computes a commitment-decommitment pair $(com, dec) = \mathrm{Com}(b_A)$ and sends com to B
- B reveals b_B
- A and B output $b = b_A \oplus b_B$.

Output b is guaranteed to be uniform in $\{0,1\}$ as long as A *or* B is honest

Zero-knowledge proofs

The identification problem: How to safely prove oneself

The identification problem

Statement: "I am the only one who knows this secret"

How can I prove that?

1. Send the secret?
   No: then the verifier also know my secret...
2. Take a signing key as secret, and show that I can sign a message?
   Still too much: the verifier learns a signature, can prove I was there, ...
3. Take a private key as secret, and show that I can decrypt a message?
   Still too much: the verifier might learn the decryption of something...

The identification problem

Statement: "I am the only one who knows this secret"

How can I prove that?

1. Send the secret?
   No: then the verifier also know my secret...
2. Take a signing key as secret, and show that I can sign a message?
   Still too much: the verifier learns a signature, can prove I was there, ...
3. Take a private key as secret, and show that I can decrypt a message?
   Still too much: the verifier might learn the decryption of something...

The identification problem

Statement: "I am the only one who knows this secret"

How can I prove that?

1. Send the secret?
   No: then the verifier also know my secret...

2. Take a signing key as secret, and show that I can sign a message?
   Still too much: the verifier learns a signature, can prove I was there, ...

3. Take a private key as secret, and show that I can decrypt a message?
   Still too much: the verifier might learn the decryption of something...

The identification problem

Statement: "I am the only one who knows this secret"

How can I prove that?

1. Send the secret?
   No: then the verifier also know my secret...
2. Take a signing key as secret, and show that I can sign a message?
   Still too much: the verifier learns a signature, can prove I was there, ...
3. Take a private key as secret, and show that I can decrypt a message?
   Still too much: the verifier might learn the decryption of something...

The identification problem

I want to prove that I am the one who knows this secret, but not to provide any other knowledge . . .

Idea: Make sure that the verifier already knows my answer!

$$
\begin{tikzcd}
P \arrow[d, Rightarrow] & & V \arrow[ll, "c = \mathrm{Enc}_{pk}(m)"'] \arrow[d, Rightarrow] \\
\bullet \arrow[rr, "com"] \arrow[d, Rightarrow] & & \bullet \arrow[d, Rightarrow] \\
\bullet \arrow[d, Rightarrow] & & \bullet \arrow[ll, "m'"'] \arrow[d, Rightarrow] \\
\bullet \arrow[rr, "dec"] & & \bullet
\end{tikzcd}
$$

- pk is P's public encryption key
- $(com, dec) \leftarrow \mathrm{Com}(m)$
- dec is sent only if $m = m'$

The identification problem

I want to prove that I am the one who knows this secret, but not to provide any other knowledge . . .

Idea: Make sure that the verifier already knows my answer!

$$\begin{array}{ccc}
P & \xleftarrow{\quad c = \mathrm{Enc}_{pk}(m) \quad} & V \\
\Downarrow & & \Downarrow \\
\bullet & \xrightarrow{\quad com \quad} & \bullet \\
\Downarrow & & \Downarrow \\
\bullet & \xleftarrow{\quad m' \quad} & \bullet \\
\Downarrow & & \Downarrow \\
\bullet & \xrightarrow{\quad dec \quad} & \bullet
\end{array}$$

- pk is P's public encryption key
- $(com, dec) \leftarrow \mathsf{Com}(m)$
- dec is sent only if $m = m'$

Proofs

- “Traditional” mathematical proofs:

  “A list of reasons that shows a statement to be true”

  - Non-interactive
  - No unique verifier in mind

- It can also be an interactive conversation
- Many applications require designated verifier proofs

Proofs

- "Traditional" mathematical proofs:

  "A list of reasons that shows a statement to be true"

  - Non-interactive
  - No unique verifier in mind

- It can also be an interactive conversation
- Many applications require designated verifier proofs

Proofs

- "Traditional" mathematical proofs:

  "A list of reasons that shows a statement to be true"

  - Non-interactive
  - No unique verifier in mind

- It can also be an interactive conversation

- Many applications require designated verifier proofs

Proofs

- "Traditional" mathematical proofs:

  "A list of reasons that shows a statement to be true"

  - Non-interactive
  - No unique verifier in mind

- It can also be an interactive conversation
- Many applications require designated verifier proofs

Interactive proofs

Three ingredients:

1. A *prover* P, possibly unbounded
2. A *verifier* V, PPT bounded
3. A language $L \subset \{0,1\}^*$ defining a set of true statements

Properties:

- Even if P is unbounded, he should not be able to prove wrong things
- V must be able to perform his task efficiently
- L can be a lot of things:
  - set of Diffie-Hellman tuples $(g, g^a, g^b, g^{ab}) \in \mathbb{G}^4$ in a cyclic group $\mathbb{G}$
  - set of pairs of isomorphic graphs

Interactive proofs

Three ingredients:

1. A *prover* P, possibly unbounded
2. A *verifier* V, PPT bounded
3. A language $L \subset \{0,1\}^*$ defining a set of true statements

Properties:

- Even if P is unbounded, he should not be able to prove wrong things
- V must be able to perform his task efficiently
- L can be a lot of things:
  - set of Diffie-Hellman tuples $(g, g^a, g^b, g^{ab}) \in \mathbb{G}^4$ in a cyclic group $\mathbb{G}$
  - set of pairs of isomorphic graphs

Interactive proofs

The pair (P, V) is an *interactive proof system* for L if:

1. **Completeness**: If $x \in L$ then the probability that P does not convince V is negligible in $|x|$
2. **Soundness**: If $x \notin L$ then the probability that any P^* convinces V is negligible in $|x|$

Observations:

- V can be convinced even if P^* is unbounded
- Proofs are probabilistic
- P may generate a proof using a *witness* w of the membership of $x \in L$ (if one exists):
  - For the set of Diffie-Hellman tuples: send a
  - For the set of isomorphic graphs: send an isomorphism

Interactive proofs

The pair (P, V) is an *interactive proof system* for L if:

1. **Completeness**: If $x \in L$ then the probability that P does not convince V is negligible in $|x|$
2. **Soundness**: If $x \notin L$ then the probability that any P^* convinces V is negligible in $|x|$

Observations:

- V can be convinced even if P^* is unbounded
- Proofs are probabilistic
- P may generate a proof using a *witness* w of the membership of $x \in L$ (if one exists):
  - For the set of Diffie-Hellman tuples: send a
  - For the set of isomorphic graphs: send an isomorphism

Interactive proofs

The pair (P, V) is an *interactive proof system* for L if:

1. **Completeness**: If $x \in L$ then the probability that P does not convince V is negligible in $|x|$
2. **Soundness**: If $x \notin L$ then the probability that any P^* convinces V is negligible in $|x|$

Observations:

- V can be convinced even if P^* is unbounded
- Proofs are probabilistic
- P may generate a proof using a *witness* w of the membership of $x \in L$ (if one exists):
  - For the set of Diffie-Hellman tuples: send a
  - For the set of isomorphic graphs: send an isomorphism

Interactive proofs

The pair (P, V) is an *interactive proof system* for L if:

1. **Completeness**: If $x \in L$ then the probability that P does not convince V is negligible in $|x|$
2. **Soundness**: If $x \notin L$ then the probability that any P^* convinces V is negligible in $|x|$

Observations:

- V can be convinced even if P^* is unbounded
- Proofs are probabilistic
- P may generate a proof using a *witness* w of the membership of $x \in L$ (if one exists):
  - For the set of Diffie-Hellman tuples: send a
  - For the set of isomorphic graphs: send an isomorphism

Interactive proofs

The pair (P, V) is an *interactive proof system* for L if:

1. **Completeness**: If $x \in L$ then the probability that P does not convince V is negligible in $|x|$
2. **Soundness**: If $x \notin L$ then the probability that any P^* convinces V is negligible in $|x|$

Observations:

- V can be convinced even if P^* is unbounded
- Proofs are probabilistic
- P may generate a proof using a *witness* w of the membership of $x \in L$ (if one exists):
  - For the set of Diffie-Hellman tuples: send a
  - For the set of isomorphic graphs: send an isomorphism

Zero-knowledge proofs

Motivation:

- Protect the prover: the verifier should not learn anything but the fact that $x \in L$; no information about w should leak

Idea:

- Let *trans* be the discussion between P and any PPT V^* on input x
- A simulator should be able to produce something indistinguishable from *trans* just from x

Observations:

- No verifier can convince that a transcript is "real": he could have produced it himself
- This "simulator" can build *trans* in any order and even rewind the verifier!

Zero-knowledge proofs

Motivation:

- Protect the prover: the verifier should not learn anything but the fact that $x \in L$; no information about w should leak

Idea:

- Let *trans* be the discussion between P and any PPT V^* on input x
- A simulator should be able to produce something indistinguishable from *trans* just from x

Observations:

- No verifier can convince that a transcript is "real": he could have produced it himself
- This "simulator" can build *trans* in any order and even rewind the verifier!

Zero-knowledge proofs

Motivation:

- Protect the prover: the verifier should not learn anything but the fact that $x \in L$; no information about w should leak

Idea:

- Let *trans* be the discussion between P and any PPT V^* on input x
- A simulator should be able to produce something indistinguishable from *trans* just from x

Observations:

- No verifier can convince that a transcript is "real": he could have produced it himself
- This "simulator" can build *trans* in any order and even rewind the verifier!

Zero-knowledge proofs

(P, V) is a *perfect zero-knowledge* interactive proof system for L if $\forall$ PPT V^*, $\exists$ a PPT simulator $\mathcal{S}_{V^*}$ s.t. $\forall \mathcal{D}$:

$$\Pr[\mathcal{D}(trans_{(P,V^*)}(x)) = 1] = \Pr[\mathcal{D}(trans_{\mathcal{S}_{V^*}}(x)) = 1]$$

where:

- $trans_{(P,V^*)}(x)$ is the transcript of the interaction of P and V^* on input x
- $trans_{\mathcal{S}_{V^*}}(x)$ is the output of $\mathcal{S}_{V^*}$ on input x
- $\mathcal{D}$ is anyone who tries to distinguish the two transcripts

Remark:

- One could define *computational zero-knowledge*:
  - $\mathcal{D}$ must be PPT
  - the probabilities can have a negligible difference

Zero-knowledge proofs

(P, V) is a *perfect zero-knowledge* interactive proof system for L if $\forall$ PPT V^*, $\exists$ a PPT simulator $\mathcal{S}_{V^*}$ s.t. $\forall \mathcal{D}$:

$$\Pr[\mathcal{D}(trans_{(P,V^*)}(x)) = 1] = \Pr[\mathcal{D}(trans_{\mathcal{S}_{V^*}}(x)) = 1]$$

where:

- $trans_{(P,V^*)}(x)$ is the transcript of the interaction of P and V^* on input x
- $trans_{\mathcal{S}_{V^*}}(x)$ is the output of $\mathcal{S}_{V^*}$ on input x
- $\mathcal{D}$ is anyone who tries to distinguish the two transcripts

Remark:

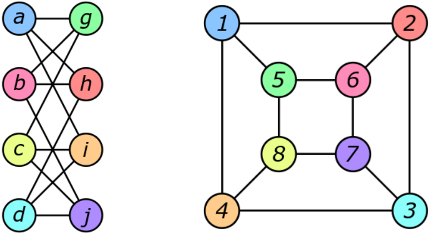
- One could define *computational zero-knowledge*:
  - $\mathcal{D}$ must be PPT
  - the probabilities can have a negligible difference

Graph isomorphism

Two graphs $G := (G_V, G_E)$ and $H := (H_V, H_E)$ are isomorphic if

- $\exists$ a bijection $f : G_V \rightarrow H_V$ and
- $(g_1, g_2) \in G_E \Leftrightarrow (f(g_1), f(g_2)) \in H_E$

Are these two graphs isomorphic?



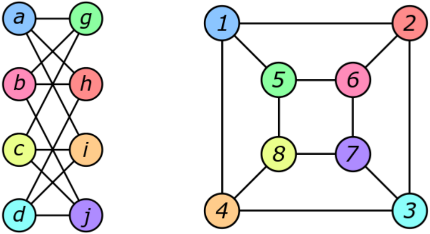
No known algorithm allows deciding in PPT whether two graphs are isomorphic

Graph isomorphism

Two graphs $G := (G_V, G_E)$ and $H := (H_V, H_E)$ are isomorphic if

- $\exists$ a bijection $f : G_V \to H_V$ and
- $(g_1, g_2) \in G_E \Leftrightarrow (f(g_1), f(g_2)) \in H_E$

Are these two graphs isomorphic?



No known algorithm allows deciding in PPT whether two graphs are isomorphic

Proof of Graph isomorphism

On input $G := (G_V, G_E)$ and $H := (H_V, H_E)$ (isomorphic):

1. P computes (or knows) a bijection $f : G_V \to H_V$
2. P repeats n times:
   a. P publishes a graph $I := (I_V, I_E)$ built as follows:
      1. select a random bijection $g : G_V \to I_V$,
      2. build I_E s.t. (G_V, G_E) and (I_V, I_E) are isomorphic

   b. V sends a random bit $c \in \{0, 1\}$ to P

   c. P answers with h where:
      - $h := g^{-1}$ if $c = 0$
      - $h := fg^{-1}$ if $c = 1$
3. V accepts the proof if, every time, h witnesses that:
   1. (I_V, I_E) is isomorphic to (G_V, G_E) when $c = 0$
   2. (I_V, I_E) is isomorphic to (H_V, H_E) when $c = 1$

Proof of Graph isomorphism

On input $G := (G_V, G_E)$ and $H := (H_V, H_E)$ (isomorphic):

1. P computes (or knows) a bijection $f : G_V \rightarrow H_V$
2. P repeats n times:
   a. P publishes a graph $I := (I_V, I_E)$ built as follows:
      1. select a random bijection $g : G_V \rightarrow I_V$,
      2. build I_E s.t. (G_V, G_E) and (I_V, I_E) are isomorphic

   b. V sends a random bit $c \in \{0, 1\}$ to P
   c. P answers with h where:
      - $h := g^{-1}$ if $c = 0$
      - $h := fg^{-1}$ if $c = 1$
3. V accepts the proof if, every time, h witnesses that:
   1. (I_V, I_E) is isomorphic to (G_V, G_E) when $c = 0$
   2. (I_V, I_E) is isomorphic to (H_V, H_E) when $c = 1$

Proof of Graph isomorphism

On input $G := (G_V, G_E)$ and $H := (H_V, H_E)$ (isomorphic):

1. P computes (or knows) a bijection $f : G_V \to H_V$
2. P repeats n times:
   a. P publishes a graph $I := (I_V, I_E)$ built as follows:
      1. select a random bijection $g : G_V \to I_V$,
      2. build I_E s.t. (G_V, G_E) and (I_V, I_E) are isomorphic
   b. V sends a random bit $c \in \{0, 1\}$ to P
   c. P answers with h where:
      - $h := g^{-1}$ if $c = 0$
      - $h := fg^{-1}$ if $c = 1$
3. V accepts the proof if, every time, h witnesses that:
   1. (I_V, I_E) is isomorphic to (G_V, G_E) when $c = 0$
   2. (I_V, I_E) is isomorphic to (H_V, H_E) when $c = 1$

Proof of Graph isomorphism

Completeness:

- P can answer all challenges

Soundness:

- If $G = (G_V, G_E)$ and $H = (H_V, H_E)$ are not isomorphic, then $I = (I_V, I_E)$ can only be isomorphic to one of them

  $\Rightarrow$ P^* has a probability $\frac{1}{2}$ of not being able to answer the challenge

- That makes a probability $\frac{1}{2^n}$ of P^* being able to convince V

Proof of Graph isomorphism

Completeness:

- P can answer all challenges

Soundness:

- If $G = (G_V, G_E)$ and $H = (H_V, H_E)$ are not isomorphic, then $I = (I_V, I_E)$ can only be isomorphic to one of them

  $\Rightarrow$ P^* has a probability $\frac{1}{2}$ of not being able to answer the challenge

- That makes a probability $\frac{1}{2^n}$ of P^* being able to convince V

Proof of Graph isomorphism

Perfect zero-knowledge: Build the simulator $\mathcal{S}_{V^*}$ as follows:

1. Start V^* and feed it with G and H
2. Repeat until $trans_{\mathcal{S}_{V^*}}$ contains n transcripts:
   a. Flip a coin $b \in_R \{0, 1\}$
   b. Build a graph I, as in the normal proof, but
      - isomorphic to G if $b = 0$
      - isomorphic to H if $b = 1$
   c. Send I to V^* and wait for $c \in \{0, 1\}$
   d. If $c = b$ then compute the permutation h that would be provided in the protocol, and append $\langle I, c, h \rangle$ to $trans_{\mathcal{S}_{V^*}}$
   e. If $c \neq b$ then rewind V^* where it was when entering this iteration and retry
3. Output $trans_{\mathcal{S}_{V^*}}$

Proof of Graph isomorphism

Perfect zero-knowledge: Build the simulator $\mathcal{S}_{V^*}$ as follows:

1. Start V^* and feed it with G and H
2. Repeat until $trans_{\mathcal{S}_{V^*}}$ contains n transcripts:
   a. Flip a coin $b \in_R \{0,1\}$
   b. Build a graph I, as in the normal proof, but
      - isomorphic to G if $b = 0$
      - isomorphic to H if $b = 1$
   c. Send I to V^* and wait for $c \in \{0,1\}$
   d. If $c = b$ then compute the permutation h that would be provided in the protocol, and append $\langle I, c, h \rangle$ to $trans_{\mathcal{S}_{V^*}}$
   e. If $c \neq b$ then rewind V^* where it was when entering this iteration and retry
3. Output $trans_{\mathcal{S}_{V^*}}$

Proof of Graph isomorphism

Observations:

- $\mathcal{S}_{V^*}$ tries to guess $c \in \{0, 1\}$, and restart/reboot V^* when it fails
- Failure probability is $\frac{1}{2}$ each time
- At each iteration, a valid transcript is obtained after n attempts, except with probability $\frac{1}{2^n}$
- If $\mathcal{S}_{V^*}$ makes n attempts at each iteration, it wins except with negligible probability $1 - n/2^n$
- If G and H are isomorphic, the simulated transcript is distributed as the real one

Σ-protocols

A family of:

- efficient,
- 3-move,
- honest-verifier zero-knowledge

protocols of the following form

Common input: P and V both have a statement x

Private input: P has a witness w showing that $x \in L$

1. P sends a *commitment* a to V
2. V sends a random *challenge* $c \in_R \{0,1\}^n$
3. P sends a *response* f

Given (a, c, f), V outputs 0 or 1

Σ-protocols

Π is a Σ-protocol for relation R if:

- It is a 3-move protocol with **completeness**, made of a *commitment* a, followed by a random *challenge* c, and ending with a *response* f
- **Special soundness:** For any pair (a, c, f) and (a, c', f') of accepting conversations on input x where $c \neq c'$, one can efficiently compute $w : (x, w) \in R$
- **Honest-verifier zero-knowledge:** There is an efficient simulator that, on input x and a challenge $c \in \{0, 1\}^n$, produces (a, f) such that (a, c, f) is distributed as in a normal proof.

Schnorr's protocol

Let $\mathbb{G}$ be a group of prime order q with generator g

$$
\begin{array}{ccc}
P & \xrightarrow{\hspace{4em} g^r \hspace{4em}} & V \\
\Downarrow & & \Downarrow \\
\bullet & \xleftarrow{\hspace{4em} c \hspace{4em}} & \bullet \\
\Downarrow & & \Downarrow \\
\bullet & \xrightarrow{\hspace{2em} r + c \cdot u \bmod q \hspace{2em}} & \bullet
\end{array}
$$

P proves knowledge of $u \in \mathbb{Z}_q$ to V who has $h = g^u \in \mathbb{G}$

1. P chooses $r \leftarrow \mathbb{Z}_q$ and commits through $a := g^r$
2. V challenges with a random $c \leftarrow \mathbb{Z}_{2^n}$
3. P responds with $f := r + c \cdot u \bmod q$
4. V accepts if $g^f = a \cdot (g^u)^c$

Schnorr's protocol

Let $\mathbb{G}$ be a group of prime order q with generator g

$$
\begin{tikzcd}
P \arrow[r, "g^r"] \arrow[d, Rightarrow] & V \arrow[d, Rightarrow] \\
\bullet \arrow[d, Rightarrow] & \bullet \arrow[l, "c"'] \arrow[d, Rightarrow] \\
\bullet \arrow[r, "r + c \cdot u \bmod q"] & \bullet
\end{tikzcd}
$$

P proves knowledge of $u \in \mathbb{Z}_q$ to V who has $h = g^u \in \mathbb{G}$

1. P chooses $r \leftarrow \mathbb{Z}_q$ and commits through $a := g^r$
2. V challenges with a random $c \leftarrow \mathbb{Z}_{2^n}$
3. P responds with $f := r + c \cdot u \bmod q$
4. V accepts if $g^f = a \cdot (g^u)^c$

Schnorr's protocol

$$\begin{tikzcd}
P \arrow[rrr, "a := g^r"] \arrow[d, Rightarrow] & & & V \arrow[d, Rightarrow] \\
\bullet \arrow[d, Rightarrow] & & & \bullet \arrow[lll, "c"'] \arrow[d, Rightarrow] \\
\bullet \arrow[rrr, "f := r + c \cdot u \bmod q"] & & & \bullet
\end{tikzcd}$$

Completeness: obvious

Soundness:

- In order to reply with non-negligible probability, P must be able to respond to more than 2 challenges, say c and c'
- Then $g^f/(g^u)^c = g^{f'}/(g^u)^{c'}$ and $u = \frac{f-f'}{c-c'}$

Honest verifier zero-knowledge:

- Given $h = g^u$ and c, choose $f \in_R \mathbb{Z}_q$ and compute $a := g^f/(g^u)^c$
  (This does not works if, say, V computes $c := \mathcal{H}(g^r)$)

Schnorr's protocol

$$\begin{array}{ccc}
P & \xrightarrow{\quad a := g^r \quad} & V \\
\Downarrow & & \Downarrow \\
\bullet & \xleftarrow{\quad c \quad} & \bullet \\
\Downarrow & & \Downarrow \\
\bullet & \xrightarrow{\quad f := r + c \cdot u \bmod q \quad} & \bullet
\end{array}$$

Completeness: obvious

Soundness:

- In order to reply with non-negligible probability, P must be able to respond to more than 2 challenges, say c and c'
- Then $g^f/(g^u)^c = g^{f'}/(g^u)^{c'}$ and $u = \frac{f-f'}{c-c'}$

Honest verifier zero-knowledge:

- Given $h = g^u$ and c, choose $f \in_R \mathbb{Z}_q$ and compute $a := g^f/(g^u)^c$ (This does not works if, say, V computes $c := \mathcal{H}(g^r)$)

The Guillou-Quisquater protocol

Let $N = pq$ be an RSA modulus and a prime e such that $\gcd(e, \varphi(N)) = 1$

$$\begin{tikzcd}
P \arrow[rr, "a := r^e \bmod N"] \arrow[d, Rightarrow] & & V \arrow[d, Rightarrow] \\
\bullet \arrow[d, Rightarrow] & & \bullet \arrow[ll, "c"'] \arrow[d, Rightarrow] \\
\bullet \arrow[rr, "f := r \cdot u^c \bmod N"] & & \bullet
\end{tikzcd}$$

P proves knowledge of $u \in \mathbb{Z}_N^*$, where $I = u^e \bmod N$ is public

1. P chooses $r \leftarrow \mathbb{Z}_N^*$ and commits through $a := r^e \bmod N$
2. V challenges with a random $c \leftarrow \{0, \ldots, e-1\}$
3. P responds with $f := r \cdot u^c \bmod N$
4. V accepts if $f^e \equiv a \cdot I^c \pmod{N}$

The Guillou-Quisquater protocol

Let $N = pq$ be an RSA modulus and a prime e such that $\gcd(e, \varphi(N)) = 1$

$$\begin{tikzcd}
P \arrow[rr, "a := r^e \bmod N"] \arrow[d, Rightarrow] & & V \arrow[d, Rightarrow] \\
\bullet \arrow[d, Rightarrow] & & \bullet \arrow[ll, "c"'] \arrow[d, Rightarrow] \\
\bullet \arrow[rr, "f := r \cdot u^c \bmod N"] & & \bullet
\end{tikzcd}$$

P proves knowledge of $u \in \mathbb{Z}_N^*$, where $I = u^e \bmod N$ is public

1. P chooses $r \leftarrow \mathbb{Z}_N^*$ and commits through $a := r^e \bmod N$
2. V challenges with a random $c \leftarrow \{0, \ldots, e-1\}$
3. P responds with $f := r \cdot u^c \bmod N$
4. V accepts if $f^e \equiv a \cdot I^c \pmod{N}$

The Guillou-Quisquater protocol

Exercises:

- Show the soundness property of GQ
  (hint: use the binding property of the RSA-based commitment)
- Show that Schnorr and GQ with binary challenges $c \in \{0, 1\}$ are perfectly ZK
- Show that any Σ protocol implies a commitment

Non-interactive ZK

Honest verifier ZK gives non-interactive proofs

Let $\mathcal{H}$ be a random oracle:

- Compute $c := \mathcal{H}(a, x)$ and send *non-interactive* proof (a, c, f)!
- Implies a signature scheme via the Fiat-Shamir heuristic

  By including the message m in the statement $c := \mathcal{H}(a, (x, m))$

The resulting protocol is sound in the ROM. Sketch:

- $\mathcal{S}$ starts P^*, answers $\mathcal{H}(a, x)$ requests with random c until it gets a valid (a, c, f) from P^*.
- Then $\mathcal{S}$ restarts P^* and answers $\mathcal{H}(a, x)$ requests with random c' until it gets a different proof for the same (a, x).

Non-interactive ZK

Honest verifier ZK gives non-interactive proofs

Let $\mathcal{H}$ be a random oracle:

- Compute $c := \mathcal{H}(a, x)$ and send *non-interactive* proof (a, c, f)!
- Implies a signature scheme via the Fiat-Shamir heuristic

  By including the message m in the statement $c := \mathcal{H}(a, (x, m))$

The resulting protocol is sound in the ROM. Sketch:

- $\mathcal{S}$ starts P^*, answers $\mathcal{H}(a, x)$ requests with random c until it gets a valid (a, c, f) from P^*.
- Then $\mathcal{S}$ restarts P^* and answers $\mathcal{H}(a, x)$ requests with random c' until it gets a different proof for the same (a, x).

Non-interactive ZK

Honest verifier ZK gives non-interactive proofs

Let $\mathcal{H}$ be a random oracle:

- Compute $c := \mathcal{H}(a, x)$ and send *non-interactive* proof (a, c, f)!
- Implies a signature scheme via the Fiat-Shamir heuristic

  By including the message m in the statement $c := \mathcal{H}(a, (x, m))$

The resulting protocol is sound in the ROM. Sketch:

- $\mathcal{S}$ starts P^*, answers $\mathcal{H}(a, x)$ requests with random c until it gets a valid (a, c, f) from P^*.
- Then $\mathcal{S}$ restarts P^* and answers $\mathcal{H}(a, x)$ requests with random c' until it gets a different proof for the same (a, x).

Proving statements about ElGamal ciphertexts

ElGamal encryption in prime-order group:

- Public key: $(g, h) := (g, g^v)$
- Ciphertext: $(c_1, c_2) := (g^u, m \cdot g^{uv})$

Statement:

- (c_1, c_2) is an encryption of m under (g, h)
- $(g, h, c_1, c_2/m) = (g, g^u, g^v, g^{uv})$ is a Diffie-Hellman tuple
- witness: either x or y

Reformulation: L contains all (g_1, g_2, g_3, g_4) s.t. $\log_{g_1}(g_2) = \log_{g_3}(g_4)$

- Either $(g_1, g_2, g_3, g_4) := (g, g^u, g^v, g^{uv})$ (witness is u)
- Or $(g_1, g_2, g_3, g_4) := (g, g^u, g^v, g^{uv})$ (witness is v)

Proving statements about ElGamal ciphertexts

ElGamal encryption in prime-order group:

- Public key: $(g, h) := (g, g^v)$
- Ciphertext: $(c_1, c_2) := (g^u, m \cdot g^{uv})$

Statement:

- (c_1, c_2) is an encryption of m under (g, h)
- $(g, h, c_1, c_2/m) = (g, g^u, g^v, g^{uv})$ is a Diffie-Hellman tuple
- witness: either x or y

Reformulation: L contains all (g_1, g_2, g_3, g_4) s.t. $\log_{g_1}(g_2) = \log_{g_3}(g_4)$

- Either $(g_1, g_2, g_3, g_4) := (g, g^u, g^v, g^{uv})$ (witness is u)
- Or $(g_1, g_2, g_3, g_4) := (g, g^u, g^v, g^{uv})$ (witness is v)

Proving statements about ElGamal ciphertexts

ElGamal encryption in prime-order group:

- Public key: $(g, h) := (g, g^v)$
- Ciphertext: $(c_1, c_2) := (g^u, m \cdot g^{uv})$

Statement:

- (c_1, c_2) is an encryption of m under (g, h)
- $(g, h, c_1, c_2/m) = (g, g^u, g^v, g^{uv})$ is a Diffie-Hellman tuple
- witness: either x or y

Reformulation: L contains all (g_1, g_2, g_3, g_4) s.t. $\log_{g_1}(g_2) = \log_{g_3}(g_4)$

- Either $(g_1, g_2, g_3, g_4) := (g, g^u, g^v, g^{uv})$ (witness is u)
- Or $(g_1, g_2, g_3, g_4) := (g, g^u, g^v, g^{uv})$ (witness is v)

Chaum-Pedersen protocol

Let $\mathbb{G}$ be a group of prime order q with generator g

$$\begin{tikzcd}
P \arrow[rr, "{g_1^r, g_3^r}"] \arrow[d, Rightarrow] & & V \arrow[d, Rightarrow] \\
\bullet \arrow[d, Rightarrow] & & \bullet \arrow[ll, "c"'] \arrow[d, Rightarrow] \\
\bullet \arrow[rr, "r + c \cdot u \bmod q"] & & \bullet
\end{tikzcd}$$

P proves that $\log_{g_1}(g_2) = \log_{g_3}(g_4)(= u)$

1. P chooses $r \leftarrow \mathbb{Z}_q$ and commits through $a := (a_1, a_3) = (g_1^r, g_3^r)$
2. V challenges with a random $c \leftarrow \mathbb{Z}_{2^n}$
3. P responds with $f := r + c \cdot u \bmod q$
4. V accepts if $g_1^f = a_1 \cdot (g_2)^c$ and $g_3^f = a_3 \cdot (g_4)^c$

Chaum-Pedersen protocol

Let $\mathbb{G}$ be a group of prime order q with generator g

$$\begin{tikzcd}
P \arrow[rr, "g_1^r, g_3^r"] \arrow[d, Rightarrow] & & V \arrow[d, Rightarrow] \\
\bullet \arrow[d, Rightarrow] & & \bullet \arrow[ll, "c"'] \arrow[d, Rightarrow] \\
\bullet \arrow[rr, "r + c \cdot u \bmod q"] & & \bullet
\end{tikzcd}$$

P proves that $\log_{g_1}(g_2) = \log_{g_3}(g_4)(= u)$

1. P chooses $r \leftarrow \mathbb{Z}_q$ and commits through $a := (a_1, a_3) = (g_1^r, g_3^r)$
2. V challenges with a random $c \leftarrow \mathbb{Z}_{2^n}$
3. P responds with $f := r + c \cdot u \bmod q$
4. V accepts if $g_1^f = a_1 \cdot (g_2)^c$ and $g_3^f = a_3 \cdot (g_4)^c$

Chaum-Pedersen protocol

Let $\mathbb{G}$ be a group of prime order q with generator g

$$
\begin{array}{ccc}
P & \xrightarrow{\quad a := (g_1^r, g_3^r) \quad} & V \\
\Downarrow & & \Downarrow \\
\bullet & \xleftarrow{\quad c \quad} & \bullet \\
\Downarrow & & \Downarrow \\
\bullet & \xrightarrow{\quad f := r + c \cdot u \bmod q \quad} & \bullet
\end{array}
$$

Completeness: obvious

Soundness:

- If P can prove with $((a_1, a_3), c, f)$ and $((a_1, a_3), c', f')$ then $u = \log_{g_1}(g_2) = \log_{g_3}(g_4) = \frac{f-f'}{c-c'}$

Honest verifier zero-knowledge:

- Given c, choose $f \in_R \mathbb{Z}_q$ and compute $a_1 := g_1^f/(g_2)^c$ and $a_3 := g_3^f/(g_4)^c$

Proving OR statements

Suppose we have:

- a Σ-protocol Π_0 for proving that $x_0 \in L_0$
- a Σ-protocol Π_1 for proving that $x_1 \in L_1$

Combining proofs:

- Proving that $x_0 \in L_0 \wedge x_1 \in L_1$ is trivial
- Can we prove that $x_0 \in L_0 \vee x_1 \in L_1$?

Applications:

- I know one of the DL of $(h_1, \ldots, h_n)$ in base g (anonymous authentication)
- This is an encryption of 0 or 1 (election)

Proving OR statements

Suppose we have:

- a Σ-protocol Π_0 for proving that $x_0 \in L_0$
- a Σ-protocol Π_1 for proving that $x_1 \in L_1$

Combining proofs:

- Proving that $x_0 \in L_0 \wedge x_1 \in L_1$ is trivial
- Can we prove that $x_0 \in L_0 \vee x_1 \in L_1$?

Applications:

- I know one of the DL of $(h_1, \ldots, h_n)$ in base g (anonymous authentication)
- This is an encryption of 0 or 1 (election)

Disjunctive proofs [CDS94]

Suppose prover has $w_i : (x_i, w_i) \in R_i$ (but not w_{1-i})

1. P selects random c_{1-i} and runs $\mathcal{S}_{1-i}$ to get a proof $(a_{1-i}, c_{1-i}, f_{1-i})$
2. P selects a_i as Π_i's definition
3. P commits on (a_0, a_1) to V
4. V challenges with c
5. P computes $c_i = c \oplus c_{1-i}$ and f_i from (w_i, a_i, c_i)
6. V accepts if (a_0, c_0, f_0) and (a_1, c_1, f_1) check for Π_0 and Π_1 and $c_0 \oplus c_1 = c$

Disjunctive proofs

Let $\mathbb{G}$ be a group of prime order q with generator g

$$\begin{tikzcd}
P \arrow[r, "{a_0, a_1}"] \arrow[d, Rightarrow] & V \arrow[d, Rightarrow] \\
\bullet \arrow[d, Rightarrow] & \bullet \arrow[l, "c"'] \arrow[d, Rightarrow] \\
\bullet \arrow[r, "{f_0, f_1}"] & \bullet
\end{tikzcd}$$

Completeness: obvious

Soundness:

- P^* has to follow either Π_0 or Π_1

Honest verifier zero-knowledge:

- Choose (c_0, c_1) at random, run $\mathcal{S}_0$, $\mathcal{S}_1$ to get (a_0, c_0, f_0) and (a_1, c_1, f_1)
- Simulated transcript is $(a_0, a_1, c_0 \oplus c_1, f_0, f_1)$

Conclusions

Zero-knowledge proof systems

- I convince you that this statement is true
- This is the only thing you learn
- You cannot use my proof to convince anyone else (interactive case)

References (available online):

- Ivan Damgård and Jesper Buus Nielsen: Commitment Schemes and Zero-Knowledge Protocols
- Ivan Damgård: On Σ-protocols

*S*lides are available online:
`http://perso.ens-lyon.fr/benoit.libert/cours-ZK.pdf`