CRYPTOGRAPHIC PROTOCOLS
2018, LECTURE 6

TRAPDOOR DISCRETE LOGARITHM. PAILLIER

HELGER LIPMAA
UNIVERSITY OF TARTU, ESTONIA
UP TO NOW
UP TO NOW

- Introduction to the field
UP TO NOW

- Introduction to the field
- Simple secure computation protocols
UP TO NOW

- Introduction to the field
- Simple secure computation protocols
  - based on DDH ("**DL is hard**" family)
UP TO NOW

- Introduction to the field
- Simple secure computation protocols
  - based on DDH ("DL is hard" family)
- Can design many efficient 2-round protocols
UP TO NOW

- Introduction to the field
- Simple secure computation protocols
  - based on DDH ("DL is hard" family)
- Can design many efficient 2-round protocols
- **General problem:** Alice needs to compute DL

Sometimes can ignore by constructing "bitwise" protocols, but still...
UP TO NOW

- Introduction to the field
- Simple secure computation protocols
  - based on DDH ("DL is hard" family)
- Can design many efficient 2-round protocols
- General problem: Alice needs to compute DL
  - which is hard by assumption

Sometimes can ignore by constructing "bitwise" protocols, but still...
THIS TIME
THIS TIME

- Trapdoor discrete logarithm
THIS TIME

- Trapdoor discrete logarithm
- hard if you do not know secret key
Trapdoor discrete logarithm
- hard if you do not know secret key
- easy if you do
THIS TIME

- Trapdoor discrete logarithm
  - hard if you do not know secret key
  - easy if you do
- Paillier cryptosystem
THIS TIME

- Trapdoor discrete logarithm
  - hard if you do not know secret key
  - easy if you do
- Paillier cryptosystem
- Some protocols
This time

- Trapdoor discrete logarithm
  - hard if you do not know secret key
  - easy if you do
- Paillier cryptosystem
- Some protocols
- E-voting
**Def. Binomial coefficient:**

\[
\binom{n}{i} = \frac{n!}{i!(n-i)!}
\]
Def. Binomial coefficient:\n\[ \binom{n}{i} = \frac{n!}{i!(n-i)!} \]

Relation with exponentiation:\n\[ (a + b)^m = \sum_{i=0}^{m} \binom{m}{i} a^i b^{m-i} \]
Def. Binomial coefficient:
\[
\binom{n}{i} = \frac{n!}{i!(n-i)!}
\]

Relation with exponentiation:
\[
(a + b)^m = \sum_{i=0}^{m} \binom{m}{i} a^i b^{m-i}
\]

\[a = n, \ b = 1:\]
\[
(n + 1)^m = \sum_{i=0}^{m} \binom{m}{i} n^i = \binom{m}{0} + \binom{m}{1} n + \binom{m}{2} n^2 + \ldots
\]
**Def. Binomial coefficient:**
\[
\binom{n}{i} = \frac{n!}{i!(n-i)!}
\]

**Relation with exponentiation:**
\[
(a + b)^m = \sum_{i=0}^{m} \binom{m}{i} a^i b^{m-i}
\]

**a = n, b = 1:**
\[
(n + 1)^m = \sum_{i=0}^{m} \binom{m}{i} n^i = \binom{m}{0} n + \binom{m}{1} n + \binom{m}{2} n^2 + \ldots
\]

**modulo n^2:**
\[
(n + 1)^m \equiv 1 + mn \pmod{n^2}
\]
**Def.** Binomial coefficient:
\[
\binom{n}{i} = \frac{n!}{i!(n-i)!}
\]

**Relation with exponentiation:**
\[
(a + b)^m = \sum_{i=0}^{m} \binom{m}{i} a^i b^{m-i}
\]

\[\text{a = n, b = 1:} \quad (n + 1)^m = \sum_{i=0}^{m} \binom{m}{i} n^i = \binom{m}{0} + \binom{m}{1} n + \binom{m}{2} n^2 + \ldots\]

**modulo n^2:**
\[
(n + 1)^m \equiv 1 + mn \pmod{n^2}
\]

Idea: while encrypting, use \(g = 1 + n\) as a generator in a group modulo \(n^2\). Needed: can compute DL only while knowing some secret
GETTING CLOSER TO PAILLIER...
GETTING CLOSER TO PAILLIER...

\[ \text{Enc}(m; r) = ((n + 1)^m \cdot hr, (n + 1)^r) \mod n^2 \] // Analogy to Elgamal
GETTING CLOSER TO PAILLIER...

- $\text{Enc}(m; r) = ((n + 1)^m h^r, (n + 1)^r) \mod n^2$ // Analogy to Elgamal
  - possible but not so efficient
**GETTING CLOSER TO PAILLIER...**

- $\text{Enc}(m; r) = ((n + 1)^m h^r, (n + 1)r) \mod n^2$ // Analogy to Elgamal

- possible but not so efficient

- $|\text{Enc}(m; r)| = 4|m|$ since $m \in \mathbb{Z}_n$

- We need later $(1 + mn) < n^2$
GETTING CLOSER TO PAILLIER...

- \( \text{Enc}(m; r) = ((n + 1)^m h^r, (n + 1)r) \mod n^2 \) // Analogy to Elgamal

- possible but not so efficient

- \(|\text{Enc}(m; r)| = 4|m| \) since \( m \in \mathbb{Z}_n \)

- Paillier cryptosystem (1999):

We need later \((1 + mn) < n^2\)
GETTING CLOSER TO PAILLIER...

- \( \text{Enc}(m; r) = ((n + 1)^m h^r, (n + 1)r) \mod n^2 \) // Analogy to Elgamal

  - possible but not so efficient

  - \( |\text{Enc}(m; r)| = 4|m| \) since \( m \in \mathbb{Z}_n \)

- **Paillier cryptosystem (1999):**

  - \( \text{Enc}(m; r) = (n + 1)^m r^m = (mn + 1) r^m \mod n^2 \)

We need later \((1 + mn) < n^2\)
GETTING CLOSER TO PAILLIER...

- $\text{Enc}(m; r) = ((n + 1)^m h^r, (n + 1)r) \mod n^2$ // Analogy to Elgamal
  - possible but not so efficient
  - $|\text{Enc}(m; r)| = 4 |m|$ since $m \in \mathbb{Z}_n$

Paillier cryptosystem (1999):

- $\text{Enc}(m; r) = (n + 1)^m r^n = (mn + 1) r^n \mod n^2$

- Trapdoor (idea):

We need later $(1 + mn) < n^2$
GETTING CLOSER TO PAILLIER...

- $\text{Enc}(m; r) = ((n + 1)^m h^r, (n + 1)^r) \mod n^2$ // Analogy to Elgamal

  - possible but not so efficient

  - $|\text{Enc}(m; r)| = 4|m|$ since $m \in \mathbb{Z}_n$

- Paillier cryptosystem (1999):

  - $\text{Enc}(m; r) = (n + 1)^m r^m = (mn + 1) r^n \mod n^2$

- Trapdoor (idea):

  - related to knowledge of factorization of $n$

We need later $(1 + mn) < n^2$
SOME ASSUMPTIONS

- Factoring
- RSA
- Strong RSA
- DCRA
- Discrete Log
- CDH
- DDH
- Various pairing assumptions
- SVP
- gapSVP
- CVP
- RLWE
- LWE
FACTORIZING
FACTORING

- **Assumption:** given a large composite number, it is infeasible to factor it
FACTORIZING

- **Assumption:** given a large composite number, it is infeasible to factor it

- Not quite: it is easy to factor any even number
FACTORING

Assumption: given a large composite number, it is infeasible to factor it

Not quite: it is easy to factor any even number

also say any square numbers
FACTORING

❖ **Assumption:** given a large composite number, it is infeasible to factor it

❖ Not quite: it is easy to factor any even number

❖ also say any square numbers

❖ **Common version:** given $n = pq$, for two **random** large primes $p$ and $q$, it is difficult to find $p$ and $q$
FACTORIZING
FACTORIZING

❖ Probably the best known hard problem
FACTORING

❖ Probably the best known hard problem

❖ The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic.

-------- Carl Friedrich Gauss
FACTORING

❖ Probably the best known hard problem

❖ The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic.

~~~~~~~ Carl Friedrich Gauss

❖ Classical computers: subexponential but superpolynomial time (like DL instantiation 1)
FACTORYING

- Probably the best known hard problem

- The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic.

- Classical computers: subexponential but superpolynomial time (like DL instantiation 1)

- Quantum computers: polynomial time (like DL)
GETTING CLOSER TO PAILLIER...
Getting closer to Paillier...

- Prime \( p \) is **safe** if \( p = 2p' + 1 \) for a prime \( p' \)
Prime $p$ is safe if $p = 2p' + 1$ for a prime $p'$

Paillier cryptosystem (1999):
Prime $p$ is safe if $p = 2p' + 1$ for a prime $p'$

Paillier cryptosystem (1999):

- for random large safe primes $p$ and $q$, let $pk = n \leftarrow pq$
Prime $p$ is safe if $p = 2p' + 1$ for a prime $p'$

Paillier cryptosystem (1999):

- for random large safe primes $p$ and $q$, let $pk = n ← pq$

We will not explain here why $p, q$ need to be safe (“safer”, and some formulas become nice)
 Cv GETTING CLOSER TO PAILLIER...

❖ Prime \( p \) is **safe** if \( p = 2p' + 1 \) for a prime \( p' \)

❖ **Paillier cryptosystem (1999):**

❖ for random large safe primes \( p \) and \( q \), let \( pk = n \leftarrow pq \)

We will not explain here why \( p, q \) need to be safe (“safer”, and some formulas become nice)

❖ \( c = Enc_n (m; r) = (mn + 1) r^n \mod n^2 \)
Getting closer to Paillier...

- Prime $p$ is **safe** if $p = 2p' + 1$ for a prime $p'$
- **Paillier cryptosystem (1999):**
  - for random large safe primes $p$ and $q$, let $pk = n \leftarrow pq$
  - $c = Enc_n(m; r) = (mn + 1) r^n \mod n^2$
  - Trapdoor (idea): knowledge of the factoring of $n$

We will not explain here why $p, q$ need to be safe (“safer”, and some formulas become nice)
Prime $p$ is **safe** if $p = 2p' + 1$ for a prime $p'$. 

**Paillier cryptosystem (1999):**

- for random large safe primes $p$ and $q$, let $pk = n \leftarrow pq$

  We will not explain here why $p, q$ need to be safe (“safer”, and some formulas become nice)

- $c = Enc_n (m; r) = (mn + 1) r^n \mod n^2$

** Trapdoor (idea):** knowledge of the factoring of $n$

**More precisely:** trapdoor = $i$, such that one can recover $m$ efficiently from $c^i \mod n^2$ only if $i$ is known
NOTE ON RANDOMIZER
Let $r = a + bn$ for $a, b \in \mathbb{Z}_n$
Let \( r = a + bn \) for \( a, b \in \mathbb{Z}_n \)

\[
\begin{align*}
\quad r^n &= (a + bn)^n = a^n + n \cdot (a^{n-1} bn) + \ldots \equiv a^n \mod n^2
\end{align*}
\]
NOTE ON RANDOMIZER

- Let \( r = a + bn \) for \( a, b \in \mathbb{Z}_n \)
- \( r^n = (a + bn)^n = a^n + n \cdot (a^{n-1} \cdot bn) + \ldots \equiv a^n \mod n^2 \)
- Thus one can always assume \( b = 0 \)
Let \( r = a + bn \) for \( a, b \in \mathbb{Z}_n \)

\[ r^n = (a + bn)^n = a^n + n \cdot (a^{n-1} bn) + \ldots \equiv a^n \mod n^2 \]

Thus one can always assume \( b = 0 \)

Or, that: \( r \in \mathbb{Z}_n^* \)
GETTING CLOSER TO PAILLIER...
`GETTING CLOSER TO PAILLIER...

❖ **Paillier**: \( c = \text{Enc}_n (m; r) = (mn + 1) r^n \mod n^2 \)
Paillier: $c = \text{Enc}_n (m; r) = (mn + 1) r^n \mod n^2$

Idea: trapdoor = $i$, such that one can recover $m$ efficiently from $c^i \mod n^2$ only if $i$ is known
`GETTING CLOSER TO PAILLIER...

- **Paillier:** \( c = \text{Enc}_n (m; r) = (mn + 1) r^n \mod n^2 \)

- **Idea:** trapdoor = \( i \), such that one can recover \( m \) efficiently from \( c^i \mod n^2 \) only if \( i \) is known

- \( c^i = (mn + 1)^i r^{ni} = (1 + imn) r^{ni} \mod n^2 \)
Getting closer to Paillier...

Paillier: \( c = Enc_n (m; r) = (mn + 1) r^n \mod n^2 \)

Idea: trapdoor = \( i \), such that one can recover \( m \) efficiently from \( c^i \mod n^2 \) only if \( i \) is known

\( c^i = (mn + 1)^i r^{ni} = (1 + imn) r^{ni} \mod n^2 \)

Most logical: \( i \) is such that \( r^{ni} = 1 \mod n^2 \)
RECALL: TOTIENT FUNCTION
RECALL: TOTIENT FUNCTION

- \( \phi (N) \) is the order of multiplicative group \( \mathbb{Z}_N^* \)
RECALL: TOTIENT FUNCTION

- $\phi(N)$ is the order of multiplicative group $\mathbb{Z}_N^*$
- In our case: for any $r \in \mathbb{Z}_{n^2}^*$, $r\phi(n^2) \equiv 1 \mod n^2$
RECALL: TOTIENT FUNCTION

- $\phi(N)$ is the order of multiplicative group $\mathbb{Z}_N^*$

- In our case: for any $r \in \mathbb{Z}_{n^2}^*$, $r\phi(n^2) = 1 \mod n^2$

- $\phi(p) = p - 1$ for prime $p$ (0 is not invertible)
Recall: Totient Function

- \( \phi(N) \) is the order of multiplicative group \( \mathbb{Z}_N^* \)
- In our case: for any \( r \in \mathbb{Z}_{n^2}^* \), \( r \phi(n^2) = 1 \mod n^2 \)
- \( \phi(p) = p - 1 \) for prime \( p \) (0 is not invertible)
- \( \phi(N) = N \cdot \prod_{p \mid N} (1 - 1/p) \) in general
RECALL: TOTIEN T FUNCTION

- $\phi(N)$ is the **order** of multiplicative group $\mathbb{Z}_N^*$
- In our case: for any $r \in \mathbb{Z}_{n^2}^*$, $r\phi(n^2) = 1 \mod n^2$
- $\phi(p) = p - 1$ for prime $p$ (0 is not invertible)
- $\phi(N) = N \cdot \prod_{p|N} (1 - 1/p)$ in general
  - product over all distinct primes $p$ that divide $N$
RECALL: TOTIENT FUNCTION

- \( \phi(N) \) is the **order** of multiplicative group \( \mathbb{Z}_N^* \)
- In our case: for any \( r \in \mathbb{Z}_{n^2}^* \), \( r\phi(n^2) = 1 \mod n^2 \)
- \( \phi(p) = p - 1 \) for prime \( p \) (0 is not invertible)
- \( \phi(N) = N \cdot \prod_{p | N} (1 - 1/p) \) in general
  - product over all distinct primes \( p \) that divide \( N \)
- **Here:** \( \phi(n^2) = p^2q^2(1 - 1/p)(1 - 1/q) = pq(p - 1)(q - 1) = n \cdot \phi(n) \)
RECALL: TOTIENT FUNCTION

- $\phi (N)$ is the order of multiplicative group $\mathbb{Z}_N^*$
- In our case: for any $r \in \mathbb{Z}_{n^2}^*$, $r\phi (n^2) = 1 \mod n^2$
- $\phi (p) = p - 1$ for prime $p$ (0 is not invertible)
- $\phi (N) = N \cdot \prod_{p \mid N} (1 - 1/p)$ in general
  - product over all distinct primes $p$ that divide $N$
- Here: $\phi (n^2) = p^2q^2 (1 - 1/p) (1 - 1/q) = pq(p - 1)(q - 1) = n \cdot \phi (n)$
- $p = 2p' + 1$, $q = 2q' + 1$ are safe: $\phi(n) = (p - 1)(q - 1) = 4 p'q'$
FACTORING => Φ
FACTORIZING => Φ

✧ Assume knowledge of $p, q$
FACTORING \Rightarrow \Phi

- Assume knowledge of $p, q$
- Can efficiently compute $\phi := \phi(n) = (p - 1)(q - 1)$
QUIZ: $\Phi \Rightarrow$ FACTORING
QUIZ: $\Phi \Rightarrow$ FACTORING

Assume knowl. of $n=pq$ and $\phi=(p-1)(q-1)=n-(p+q)+1$
QUICK: Φ => FACTORING

- Assume knowl. of \( n=pq \) and \( \phi=(p-1)(q-1)=n-(p+q)+1 \)
- Knowing \( n \) and \( \phi \), one can compute \( s=p+q \)
QUIZ: $\Phi \implies$ FACTORING

- Assume knowl. of $n = pq$ and $\phi = (p-1)(q-1) = n - (p+q) + 1$

- Knowing $n$ and $\phi$, one can compute $s = p + q$

- Knowing $n = pq$ and $s$:
QUIZ: Φ => FACTORING

- Assume knowl. of \( n = pq \) and \( \phi = (p-1)(q-1) = n - (p+q) + 1 \)
  - Knowing \( n \) and \( \phi \), one can compute \( s = p + q \)

- Knowing \( n = pq \) and \( s \):
  - \( q = s - p \), thus \( n = p (s - p) \), thus \( p^2 - sp + n = 0 \)
**QUIZ: \( \Phi \Rightarrow \text{FACTORYING} \)**

- Assume knowl. of \( n=pq \) and \( \phi=(p-1)(q-1)=n-(p+q)+1 \)

- Knowing \( n \) and \( \phi \), one can compute \( s = p + q \)

- Knowing \( n = pq \) and \( s \):
  - \( q = s - p \), thus \( n = p(s - p) \), thus \( p^2 - sp + n = 0 \)

- Quadratic equation, thus can find \( p \) efficiently
GETTING CLOSER TO PAILLIER...
Paillier: \( c = \text{Enc}(m; r) = (mn + 1) r^n \mod n^2 \)
GETTING CLOSER TO PAILLIER...

dać Paillier: $c = \text{Enc}(m; r) = (mn + 1) r^n \mod n^2$

 difíc Trapdoor: $\phi := (\phi - 1) (q - 1), \mu := \phi^{-1} \mod n$
**GETTING CLOSER TO PAILLIER...**

- **Paillier:** \( c = \text{Enc}(m; r) = (mn + 1) r^n \mod n^2 \)

- **Trapdoor:** \( \phi := (p - 1) (q - 1), \mu := \phi^{-1} \mod n \)

- **Thus** \( (r^n)\phi = r^{\phi(n^2)} = 1 \mod n^2 \)
**GETTING CLOSER TO PAILLIER...**

- **Paillier:** $c = \text{Enc}(m; r) = (mn + 1) \cdot r^n \mod n^2$

- **Trapdoor:** $\phi := (p - 1)(q - 1)$, $\mu := \phi^{-1} \mod n$

- Thus $(r^n)^\phi = r^{\phi(n^2)} = 1 \mod n^2$

- Thus $c^\phi = (mn + 1)^\phi = \phi mn + 1 \mod n^2$
GETTING CLOSER TO PAILLIER...
GETTING CLOSER TO PAILLIER...

\[ c^\phi = (mn + 1)^\phi = \phi mn + 1 \mod n^2 \]
GETTING CLOSER TO PAILLIER...

- \( c^\phi = (mn + 1)^\phi = \phi mn + 1 \mod n^2 \)
- **Decryption:** // Need to recover \( m \) from \( c^\phi \)
**GETTING CLOSER TO PAILLIER...**

- \( c^\phi = (mn + 1)^\phi = \phi mn + 1 \mod n^2 \)

  **Decryption:** // Need to recover \( m \) from \( c^\phi \)

  Define \( L(x) := (x - 1) / n \) for \( x < n^2 \)

  **Problem:** \( n \) is not invertible modulo \( n^2 \)

Assuming \( x < n^2 \), this is integer division; means \( m < n \)
\[ c^\phi = (mn + 1)^\phi = \phi mn + 1 \mod n^2 \]

**Decryption:** // Need to recover \( m \) from \( c^\phi \)

- Define \( L(x) := (x - 1) / n \) for \( x < n^2 \)
- **Problem:** \( n \) is not invertible modulo \( n^2 \)
- Thus \( L(c^\phi \mod n^2) = \phi m \mod n \)

Assuming \( x < n^2 \), this is integer division; means \( m < n \).
\[ c^\phi = (mn + 1)\phi = \phi mn + 1 \mod n^2 \]

**Decryption:** // Need to recover \( m \) from \( c^\phi \)

- Define \( L(x) := (x - 1) / n \) for \( x < n^2 \)
- **Problem:** \( n \) is not invertible modulo \( n^2 \)
- Thus \( L(c^\phi \mod n^2) = \phi m \mod n \)
- \( m \leftarrow L(c^\phi \mod n^2) \cdot \mu \mod n \)

Assuming \( x < n^2 \), this is integer division; means \( m < n \)
GETTING CLOSER TO PAILLIER...

\[ c^\phi = (mn + 1)^\phi = \phi mn + 1 \mod n^2 \]

**Decryption:** // Need to recover \( m \) from \( c^\phi \)

- Define \( L(x) := (x - 1) / n \) for \( x < n^2 \)
- **Problem:** \( n \) is not invertible modulo \( n^2 \)
- Thus \( L(c^\phi \mod n^2) = \phi m \mod n \)
- \( m \leftarrow L(c^\phi \mod n^2) \cdot \mu \mod n \)
- \( = \phi m \phi^{-1} = m \mod n \)

Assuming \( x < n^2 \), this is integer division; means \( m < n \)
PAILLIER ENCRYPTION
PAILLIER ENCRYPTION

Sample $p, q$
\[ n \leftarrow pq \]
\[ \phi \leftarrow (p-1)(q-1) \]
\[ \mu \leftarrow \phi^{-1} \mod n \]
PAILLIER ENCRYPTION

public key $n$

Sample $p, q$

$n \leftarrow pq$

$\phi \leftarrow (p-1)(q-1)$

$\mu \leftarrow \phi^{-1} \text{mod } n$
PAILLIER ENCRYPTION

$r \in \mathbb{Z}_n^*, m \in \mathbb{Z}_n$,

Sample $p, q$

$n \leftarrow pq$

$\phi \leftarrow (p-1)(q-1)$

$\mu \leftarrow \phi^{-1} \mod n$

public key $n$
PAILLIER ENCRYPTION

$r \in \mathbb{Z}^*_n, m \in \mathbb{Z}_n$

public key $n$

$c \leftarrow (mn + 1) r^n \mod n^2$

Sample $p, q$

$n \leftarrow pq$

$\phi \leftarrow (p-1)(q-1)$

$\mu \leftarrow \phi^{-1} \mod n$
PAILLIER ENCRYPTION

$r \in \mathbb{Z}_n^*, m \in \mathbb{Z}_n$

public key $n$

$c \leftarrow (mn + 1) r^n \mod n^2$

$c^* \leftarrow c^\phi \mod n^2$

$m \leftarrow L(c^*) \cdot \mu \mod n$

Sample $p, q$

$n \leftarrow pq$

$\phi \leftarrow (p-1)(q-1)$

$\mu \leftarrow \phi^{-1} \mod n$
PAILLIER ENCRYPTION

Paillier.Setup (rκ):
1. Choose good keylength
   • = length of p, q
2. Return gk ← keylength

\[ r \in \mathbb{Z}_n^*, m \in \mathbb{Z}_n \]

Sample p, q
\[ n \leftarrow pq \]
\[ \phi \leftarrow (p-1)(q-1) \]
\[ \mu \leftarrow \phi^{-1} \mod n \]

\[ c \leftarrow (mn + 1) r^n \mod n^2 \]

\[ c^* \leftarrow c\phi \mod n^2 \]
\[ m \leftarrow L(c^*) \cdot \mu \mod n \]
**Paillier Encryption**

**Paillier.Setup (\(r^\kappa\))**:  
1. Choose good keylength
   - \(=\) length of \(p, q\)  
2. Return \(gk \leftarrow\) keylength

\[ r \in \mathbb{Z}_n^*, m \in \mathbb{Z}_n \]

**Paillier.Keygen (gk)**:  
1. \(p, q \leftarrow\) random keylength-long primes  
2. \(n \leftarrow pq\)  
3. \(\phi \leftarrow (p-1)(q-1); \mu \leftarrow \phi^{-1} \mod n\);  
4. Return \((sk=(\phi, \mu), pk=n)\)

\[ c \leftarrow (mn + 1) r^n \mod n^2 \]

\[ c^* \leftarrow c\phi \mod n^2 \]

\[ m \leftarrow L(c^*) \cdot \mu \mod n \]
PAILLIER ENCRYPTION

Paillier.Setup (\(\kappa\)):  
1. Choose good keylength  
   • = length of \(p, q\)  
2. Return \(gk \leftarrow\) keylength

Paillier.Keygen (gk):  
1. \(p, q \leftarrow\) random keylength-long primes  
2. \(n \leftarrow pq\)  
3. \(\phi \leftarrow (p-1)(q-1); \mu \leftarrow \phi^{-1} \mod n;\)  
4. Return (sk=(\(\phi, \mu\)), pk=n)

Paillier.Enc\(_{pk=n}\) (\(m; r\)):  
1. /* Assumes \(r \leftarrow \mathbb{Z}_n^*\): randomized alg. */  
2. \(c \leftarrow (mn + 1) \cdot r^n \mod n^2\)  
3. Return \(c\)

\[c^* \leftarrow c\phi \mod n^2\]
\[m \leftarrow L(c^*) \cdot \mu \mod n\]
**Paillier Encryption**

**Paillier.Setup** ($r^x$):
1. Choose good keylength
   - = length of $p$, $q$
2. Return $gk \leftarrow$ keylength

**Paillier.Keygen** ($gk$):
1. $p, q \leftarrow$ random keylength-long primes
2. $n \leftarrow pq$
3. $\phi \leftarrow (p-1)(q-1)$; $\mu \leftarrow \phi^{-1} \mod n$;
4. Return $(sk=(\phi, \mu), pk=n)$

**Paillier.Enc** $pk=n (m; r)$:
1. /* Assumes $r \leftarrow \mathbb{Z}_n^*$: randomized alg. */
2. $c \leftarrow (mn + 1) r^n \mod n^2$
3. Return $c$

**Paillier.Dec** $sk=(\phi, \mu) (c)$:
1. $m \leftarrow L (c^\phi \mod n^2) \cdot \mu \mod n$
2. Return $m$
SECURITY
SECURITY

- If factoring is easy then Paillier can be broken
If factoring is easy then Paillier can be broken
Opposite not known
SECURITY

- If factoring is easy then Paillier can be broken
- Opposite not known
- How would you come up with precise security assumption?
SECURITY

- If factoring is easy then Paillier can be broken
- Opposite not known
- How would you come up with precise security assumption?
- Tautological assumption... :(
If factoring is easy then Paillier can be broken

Opposite not known

How would you come up with precise security assumption?

Tautological assumption... :(  

but been well known for 20 years
IND-CPA SECURITY

Chosen-Plaintext Attack

\[ r, \beta \leftarrow \{0,1\} \]

\[ n \]

\[ (m_0, m_1) \]

\[ c \leftarrow (m_\beta n + 1) r^n \mod n^2 \]

\[ m \leftarrow L(c^\phi) \cdot \mu \mod n \]
IND-CPA SECURITY

\[ \beta \leftarrow \{0, 1\} \]

\[ (m_0, m_1) \]

\[ c \leftarrow (m_\beta n + 1) r^n \mod n^2 \]

\[ \beta^* \]

\[ \beta^* = \beta \]

\[ m \leftarrow L(c^\phi) \cdot \mu \mod n \]
IND-CPA SECURITY

\[ \beta^* = \beta \]

\[ m \leftarrow L(c^\phi) \cdot \mu \mod n \]

In tautological assumption, \( m_1 = 0 \) (like in DDH)
**DCR ASSUMPTION**

**Decisional Composite Residuosity**

- **Paillier** = (Setup, Keygen, Enc, Dec)
  \[
  \text{Adv}^{\text{DCR}}_{\text{Setup,}\mathcal{A}}(\kappa) := 2 \cdot \left| \Pr[\text{DCR}_{\text{Setup,}\mathcal{A}}(\kappa) = 1] - 1/2 \right|
  \]
- **A(\tau,\varepsilon)**-breaks DCR security iff
  \[
  \text{Adv}^{\text{DCR}}_{\text{Setup,}\mathcal{A}}(\kappa) \geq \varepsilon \text{ and } A \text{ runs in time } \leq \tau
  \]
- **DCR is (\tau,\varepsilon)**-secure iff no PPT adversary (\tau,\varepsilon)-breaks DCRA security
- **DCR is secure** iff it is (\text{poly}(\kappa),\text{negl}(\kappa))-secure

---

**Game DCR\text{Setup,}\mathcal{A}(\kappa)**

1. \(g_k \leftarrow \text{Setup}(1^\kappa)\)
2. \(((\phi,\mu), n) \leftarrow \text{Keygen}(1^\kappa)\)
3. \(m \leftarrow$ \mathbb{Z}_n\)
4. \(r \leftarrow$ \mathbb{Z}_n^*\)
5. \(c_0 \leftarrow \text{Enc}_n(m; r)\)
6. \(c_1 \leftarrow \text{Enc}_n(0; r)\)
7. \(\beta \leftarrow \{0, 1\}\)
8. \(\beta^* \leftarrow A(n, c_\beta)\)
9. Return \(\beta = \beta^* \oplus 1 : 0\)
**Theorem.** DCR is \((\approx \tau, \approx \varepsilon)\)-secure iff Paillier is \((\tau, \varepsilon)\)-IND-CPA secure.
**Theorem.** DCR is \((\approx \tau, \approx \varepsilon)\)-secure iff Paillier is \((\tau, \varepsilon)\)-IND-CPA secure.

**Proof idea.** DCR states it is difficult to distinguish encryption of 0 from encryption of \(m\). But then it is also difficult to distinguish encryptions of any two plaintexts.
MALLEABILITY OF PAILLIER
MALLEABILITY OF PAILLIER

Recall: $\text{Enc}_n (m; r) = (n + 1)^m r^n \mod n^2$
Recall: $\text{Enc}_n (m; r) = (n + 1)^m r^n \mod n^2$

$\Rightarrow \text{Enc}_n (m; r) \cdot \text{Enc}_n (m'; r') = \text{Enc}_n (m + m'; r \cdot r')$
MALLEABILITY OF PAILLIER

❖ Recall: $\text{Enc}_n (m; r) = (n + 1)^m r^n \mod n^2$
❖ $\Rightarrow \text{Enc}_n (m; r) \cdot \text{Enc}_n (m'; r') = \text{Enc}_n (m + m'; r \cdot r')$
❖ Thus, **additively homomorphic**
MALLEABILITY OF PAILLIER

🔹 Recall: \( \text{Enc}_n (m; r) = (n + 1)^m r^n \mod n^2 \)

🔹 \( \Rightarrow \) \( \text{Enc}_n (m; r) \cdot \text{Enc}_n (m'; r') = \text{Enc}_n (m + m'; r \cdot r') \)

🔹 Thus, **additively homomorphic**

🔹 just remember that **randomizer multiplies**
Recall: $\text{Enc}_n (m; r) = (n + 1)^m r^n \mod n^2$

$\Rightarrow \text{Enc}_n (m; r) \cdot \text{Enc}_n (m'; r') = \text{Enc}_n (m + m'; r \cdot r')$

Thus, **additively homomorphic**

just remember that **randomizer multiplies**

**Blinding:**
Malleability of Paillier

- Recall: $Enc_n(m; r) = (n + 1)^m r^n \mod n^2$
  - $\Rightarrow Enc_n(m; r) \cdot Enc_n(m'; r') = Enc_n(m + m'; r \cdot r')$
- Thus, additively homomorphic
- Just remember that randomizer multiplies
- Blinding:
  - $Enc_n(m; r) Enc_n(0; r') = Enc_n(m; rr')$
EFFICIENCY
EFFICIENCY

- Factoring of $n$ must be hard
EFFICIENCY

- Factoring of $n$ must be hard
- Thus $|n| \geq 2000$
EFFICIENCY

- Factoring of $n$ must be hard
- Thus $|n| \geq 2000$
- $(|n^2| = 2|n| = 4000)$-bit arithmetic
EFFICIENCY

- Factoring of $n$ must be hard
  - Thus $|n| \geq 2000$
  - $(|n^2| = 2|n| = 4000)$-bit arithmetic
  - vs 256-bit arithmetic with Elgamal+elliptic curves
EFFICIENCY

❖ Factoring of $n$ must be hard
❖ Thus $|n| \geq 2000$
❖ $(|n^2| = 2|n| = 4000)$-bit arithmetic
❖ vs 256-bit arithmetic with Elgamal+elliptic curves
❖ Much, much slower due to different bitlengths...
EFFICIENCY

❖ Factoring of $n$ must be hard
❖ Thus $|n| \geq 2000$
❖ $(|n^2| = 2|n| = 4000)$-bit arithmetic
❖ vs 256-bit arithmetic with Elgamal+elliptic curves
❖ Much, much slower due to different bitlengths...
❖ but: decryption does not need computation of DL
EFFICIENCY

- Factoring of $n$ must be hard
  - Thus $|n| \geq 2000$
  - $(|n^2| = 2|n| = 4000)$-bit arithmetic
  - vs $256$-bit arithmetic with Elgamal+elliptic curves
- Much, much slower due to different bitlengths...
  - but: decryption does not need computation of DL
  - $n$-bit arithmetic on elliptic curves is slower and more difficult to implement than $n$-bit modular arithmetic
(2, 1)-CPIR PROTOCOL

\[pk, sk \]
\[x \in \{0, 1\}\]

\[f = (f_0, f_1)\]
\[f_i \in \{0, 1\}^L\]
$(2, 1)$-CPIR Protocol

- $pk, sk$
- $x \in \{0, 1\}$
- $r \leftarrow \mathbb{Z}_n^*$
- $c \leftarrow \text{Enc}(x; r)$
- $f = (f_0, f_1)$
- $f_i \in \{0, 1\}^L$
(2, 1)-CPIR PROTOCOL

\[ \mathcal{c} \leftarrow \mathbb{Z}_n^* \]

\[ \mathcal{c} \leftarrow \text{Enc}(x; r) \]

\[ x \in \{0, 1\} \]

\[ \mathcal{f} = (f_0, f_1) \]

\[ f_i \in \{0, 1\}^L \]
(2, 1)-CPIR PROTOCOL

\[ (2, 1) \text{-CPIR PROTOCOL} \]

\[ \text{pk, sk} \]

\[ x \in \{0, 1\} \]

\[ r \leftarrow \mathbb{Z}_n^* \]

\[ c \leftarrow \text{Enc} (x; r) \]

\[ r' \leftarrow \mathbb{Z}_n^* \]

\[ d \leftarrow c f_1 - f_0 \cdot \text{Enc} (f_0; r') \]
(2, 1)-CPIR PROTOCOL

\[ pk, sk \]

\[ x \in \{0, 1\} \]

\[ r \leftarrow \mathbb{Z}_n^* \]

\[ c \leftarrow \text{Enc}(x; r) \]

\[ d \leftarrow c f_1 - f_0 \cdot \text{Enc}(f_0; r') \]

\[ f = (f_0, f_1) \]

\[ f_i \in \{0, 1\}^L \]
(2, 1)-CPIR PROTOCOL

pk, sk, x ∈ \{0, 1\}

M ← Dec (d)

r ← ℤ_{n}^{*}

c ← Enc (x; r)

d ← c f_{1} - f_{0} \cdot Enc (f_{0}; r')

pk

f = (f_{0}, f_{1})

f_{i} ∈ \{0, 1\}^{L}
<table>
<thead>
<tr>
<th></th>
<th>Commun. (bits)</th>
<th>Alice's comp. (exp, DL)</th>
<th>Bob's comp. (exp)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Elgamal</strong></td>
<td>4 \cdot 256</td>
<td>3 \cdot 256^{2.58} + 2^{L/2}</td>
<td>5 \cdot 256^{2.58}</td>
</tr>
<tr>
<td><strong>Elgamal bitwise</strong></td>
<td>(2L + 2) \cdot 256</td>
<td>(L + 2) \cdot 256^{2.58}</td>
<td>2L \cdot 256^{2.58}</td>
</tr>
<tr>
<td><strong>Paillier</strong></td>
<td>4000</td>
<td>2 \cdot 4000^{2.58}</td>
<td>2 \cdot 4000^{2.58}</td>
</tr>
</tbody>
</table>

- Concrete keylengths may vary, omitted constants, etc
- For first protocol, need $L \leq 256$
- For third protocol, need $L \leq 2000$
- For larger $L$, apply protocol many times for 256-bit chunks
## CPIR Complexity

<table>
<thead>
<tr>
<th></th>
<th>Commun. (bits)</th>
<th>Alice's comp. (exp, DL)</th>
<th>Bob's comp. (exp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elgamal</td>
<td>$4 \cdot 256$</td>
<td>$3 \cdot 256^{2.58} + 2^{L/2}$</td>
<td>$5 \cdot 256^{2.58}$</td>
</tr>
<tr>
<td>Elgamal bitwise</td>
<td>$(2L + 2) \cdot 256$</td>
<td>$(L + 2) \cdot 256^{2.58}$</td>
<td>$2L \cdot 256^{2.58}$</td>
</tr>
<tr>
<td>Paillier</td>
<td>$4000$</td>
<td>$2 \cdot 4000^{2.58}$</td>
<td>$2 \cdot 4000^{2.58}$</td>
</tr>
</tbody>
</table>

- Concrete keylengths may vary, omitted constants, etc
- For first protocol, need $L \leq 256$
- For third protocol, need $L \leq 2000$
CPIR: ALICE COMP.

Ignoring constants, just look at the slope
CPIR: ALICE COMP.

Ignoring constants, just look at the slope
## CPIR: LESSONS

<table>
<thead>
<tr>
<th>Range of $L$</th>
<th>Best protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L &lt; 45$</td>
<td>Elgamal</td>
</tr>
<tr>
<td>$45 &lt; L &lt; 750$</td>
<td>Elgamal bitwise</td>
</tr>
<tr>
<td>$750 &lt; L$</td>
<td>Paillier</td>
</tr>
</tbody>
</table>

Cutoff points are *approximate*
**CPIR: LESSONS**

<table>
<thead>
<tr>
<th>Range of $L$</th>
<th>Best protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L &lt; 45$</td>
<td>Elgamal</td>
</tr>
<tr>
<td>$45 &lt; L &lt; 750$</td>
<td>Elgamal bitwise</td>
</tr>
<tr>
<td>$750 &lt; L$</td>
<td>Paillier</td>
</tr>
</tbody>
</table>

Cutoff points are *approximate*

Takes at least $2^{32}$ computation - not efficient with *any* cryptosystem.
### CPIR: LESSONS

<table>
<thead>
<tr>
<th>Range of $L$</th>
<th>Best protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L &lt; 45$</td>
<td>Elgamal</td>
</tr>
<tr>
<td>$45 &lt; L &lt; 750$</td>
<td>Elgamal bitwise</td>
</tr>
<tr>
<td>$750 &lt; L$</td>
<td>Paillier</td>
</tr>
</tbody>
</table>

**Lesson:** (conventional) public-key based protocols are **slow** (for this application)

**Cutoff points are approximate**

**Takes at least $2^{32}$ computation - not efficient with any cryptosystem**
## CPIR: LESSONS

<table>
<thead>
<tr>
<th>Range of $L$</th>
<th>Best protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L &lt; 45$</td>
<td>Elgamal</td>
</tr>
<tr>
<td>$45 &lt; L &lt; 750$</td>
<td>Elgamal bitwise</td>
</tr>
<tr>
<td>$750 &lt; L$</td>
<td>Paillier</td>
</tr>
</tbody>
</table>

Lesson: (conventional) public-key based protocols are **slow** (for this application)

Cutoff points are **approximate**

Takes at least $2^{32}$ computation - not efficient with **any** cryptosystem

### However, highly optimised implementations exist.
Paillier: [https://eprint.iacr.org/2015/864.pdf](https://eprint.iacr.org/2015/864.pdf)
MORE PROTOCOLS
MORE PROTOCOLS

- We only saw 2-message 2-party protocols
MORE PROTOCOLS

- We only saw 2-message 2-party protocols

- but there are many more types of protocols...
E-VOTING: MOTIVATION
E-VOTING: MOTIVATION
E-VOTING: MOTIVATION
E-VOTING: MOTIVATION
E-VOTING: MOTIVATION
E-VOTING: MOTIVATION

Motivations:
- better security
- direct democracy
- conveniency
E-VOTING

$c_i$ \rightarrow \text{User computer} \rightarrow \text{Enc}(c_i) \rightarrow \text{Tally}(\{c_i\})
TWO-CANDIDATE E-VOTING

Vote collector: sees who sent which ciphertext, **cannot decrypt**

Tallier: sees **anonymous** ciphertexts, can decrypt

\[ c_i \in \{0, 1\} \]
TWO-CANDIDATE E-VOTING

Vote collector: sees who sent which ciphertext, cannot decrypt

Tallier: sees anonymous ciphertexts, can decrypt

$c_i \in \{0, 1\}$
TWO-CANDIDATE E-VOTING

$c_i \in \{0, 1\}$

Vote collector: sees who sent which ciphertext, **cannot** decrypt

Tallier: sees **anonymous** ciphertexts, can decrypt
TWO-CANDIDATE E-VOTING

$c_i \in \{0, 1\}$

$\text{Vote collector: sees who sent which ciphertext, cannot decrypt}$

$\text{Tallier: sees anonymous ciphertexts, can decrypt}$

$\text{pk}$

$\text{Enc}(c_i)$

$\text{pk}$

$\text{Enc}(\Sigma c_i)$

$\text{sk}$

$\Sigma c_i$
TWO-CANDIDATE E-VOTING

$c_i \in \{0, 1\}$

$\Sigma c_i = \#\text{voters who prefered 1}$

complete tally can be computed from this efficiently
QUIZ: HOW TO?

\[ \text{Enc}\left( f(c_i) \right) \rightarrow \text{Enc}(\Sigma f(c_i)) \rightarrow g(\Sigma f(c_i)) \]

\[ c_i \in \{0, \ldots, C - 1\} \]

pk \text{ pk} \text{ sk}
QUIZ: HOW TO?

$\text{Enc}(\text{f}(c_i))$

$\text{Enc}(\Sigma \text{f}(c_i))$

$\text{g}(\Sigma \text{f}(c_i))$

$\text{Hint:}$

Let $V = \#\text{voters}$

Need $V \cdot f(i) < f(i + 1)$

$c_i \in \{0, ..., C - 1\}$
MULTIPLE VOTERS

\[ f(i) := (V + 1)^i \]
MULTIPLE VOTERS

\[ f(i) := (V + 1)^i \]

Then clearly \( V \cdot f(i) = V \cdot (V + 1)^i < (V + 1)^{i+1} = f(i + 1) \)
MULTIPLE VOTERS

\[ f(i) := (V + 1)^i \]

Then clearly

\[ V \cdot f(i) = V \cdot (V + 1)^i < (V + 1)^{i+1} = f(i + 1) \]

Voter \( i \) outputs

\[ \text{Enc} \left( (V + 1)^{c_i} \right) \]

Presentation in \((V+1)\)ary number system

\[ \begin{array}{cccc}
0 & 0 & 1 & 0 \\
\end{array} \]
**MULTIPLE VOTERS**

- \( f(i) := (V + 1)^i \)
- Then clearly \( V \cdot f(i) = V \cdot (V + 1)^i < (V + 1)^{i+1} = f(i + 1) \)
- Voter \( i \) outputs \( \text{Enc} \left( (V + 1)^{c_i} \right) \)
- Server 1 outputs \( \text{Enc} \left( \sum_i (V + 1)^{c_i} \right) \)

<table>
<thead>
<tr>
<th>( T_V )</th>
<th>...</th>
<th>( T_1 )</th>
<th>( T_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
MULTIPLE VOTERS

- \( f(i) := (V + 1)^i \)

- Then clearly \( V \cdot f(i) = V \cdot (V + 1)^i < (V + 1)^i + 1 = f(i + 1) \)

- Voter \( i \) outputs \( \text{Enc} ((V + 1)^{c_i}) \)

- Server 1 outputs \( \text{Enc} (\sum_i (V + 1)^{c_i}) \)

- Server 2 outputs \( \sum_i (V + 1)^{c_i} = \sum_e T_e (V + 1)^e \)
EFFICIENCY

- The number to be decrypted: \( \leq (V + 1)^C \)
- Country like Estonia:
  - \( V = 500\,000, \ C = 150 \Rightarrow (V + 1)^C \approx 2^{2840} \)
- Cannot compute DL of this size!
- **Must** use Paillier
AVOIDING SECURE COMPUTATION
AVOIDING SECURE COMPUTATION

- Simpler solution:
AVOIDING SECURE COMPUTATION

❖ **Simpler solution:**
  ❖ server 1 just forwards all ciphertexts (in a shuffled order) to server 2
AVOIDING SECURE COMPUTATION

- **Simpler solution:**
  - server 1 just forwards all ciphertexts (in a shuffled order) to server 2

- **Good:**
AVOIDING SECURE COMPUTATION

- **Simpler solution:**
  - server 1 just forwards all ciphertexts (in a shuffled order) to server 2

- **Good:**
  - no need for encoding (can use much more complex ballots)
AVOIDING SECURE COMPUTATION

❖ **Simpler solution:**
  ❖ server 1 just forwards all ciphertexts (in a shuffled order) to server 2

❖ **Good:**
  ❖ no need for encoding (can use much more complex ballots)
  ❖ server 1 does not need to multiply
AVOIDING SECURE COMPUTATION

- **Simpler solution:**
  - server 1 just forwards all ciphertexts (in a shuffled order) to server 2

- **Good:**
  - no need for encoding (can use much more complex ballots)
  - server 1 does not need to multiply

- **Bad (efficiency-wise):** server 2 must decrypt $V$ ciphertexts
**AVOIDING SECURE COMPUTATION**

- **Simpler solution:**
  - server 1 just forwards all ciphertexts (in a shuffled order) to server 2

- **Good:**
  - no need for encoding (can use much more complex ballots)
  - server 1 does not need to multiply

- **Bad (efficiency-wise):** server 2 must decrypt $V$ ciphertexts
  - Usually server 2 is the bottleneck

Decryption much more costly than multiplication
**AVOIDING SECURE COMPUTATION**

- **Simpler solution:**
  - server 1 just forwards all ciphertexts (in a shuffled order) to server 2

- **Good:**
  - no need for encoding (can use much more complex ballots)
  - server 1 does not need to multiply

- **Bad (efficiency-wise):** server 2 must decrypt $V$ ciphertexts
  - Usually server 2 is the bottleneck

- Complicated in malicious model
AVOIDING SECURE COMPUTATION

❖ Simpler solution:
  ❖ server 1 just forwards all ciphertexts (in a shuffled order) to server 2
❖ Good:
  ❖ no need for encoding (can use much more complex ballots)
  ❖ server 1 does not need to multiply
❖ Bad (efficiency-wise): server 2 must decrypt \( V \) ciphertexts
  ❖ Usually server 2 is the bottleneck
❖ Complicated in malicious model
  ❖ see later lectures (“mixnet”)
AVOIDING SECURE COMPUTATION

- **Simpler solution:**
  - server 1 just forwards all ciphertexts (in a shuffled order) to server 2
- **Good:**
  - no need for encoding (can use much more complex ballots)
  - server 1 does not need to multiply
- **Bad (efficiency-wise):** server 2 must decrypt $V$ ciphertexts
  - Usually server 2 is the bottleneck

- Complicated in malicious model
  - see later lectures (“mixnet”)

However, this is usually the way to go since real ballots are too complex.
STUDY OUTCOMES
STUDY OUTCOMES

- Trapdoor discrete logarithm: idea
StudY ouTcoMes

🔹 Trapdoor discrete logarithm: idea
🔹 Paillier with all gory details
STUDY OUTCOMES

- Trapdoor discrete logarithm: idea
- Paillier with all gory details
- Example: efficiency of Paillier
STUDY OUTCOMES

- Trapdoor discrete logarithm: idea
- Paillier with all gory details
- Example: efficiency of Paillier
- E-voting
WHAT NEXT?
WHAT NEXT?

- We learned how to do stuff in semihonest model
WHAT NEXT?

- We learned how to do stuff in semihonest model
- Next lectures: the real thing
WHAT NEXT?

- We learned how to do stuff in semihonest model
- Next lectures: the real thing
  - Malicious model: parties do not follow the protocol
WHAT NEXT?

- We learned how to do stuff in semihonest model
- Next lectures: the real thing
  - Malicious model: parties do not follow the protocol
- Next lecture: a bit philosophical
WHAT NEXT?

❖ We learned how to do stuff in semihonest model

❖ Next lectures: the real thing

❖ Malicious model: parties do not follow the protocol

❖ Next lecture: a bit philosophical

❖ Note: reordering of lectures compared to 2016