Department of
Computer Science and Engineering
计算机科学与工程系

CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

Dr. Qi Wang

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

Protocol QR

- **Recall** If n is an integer, then $x \in \mathbb{Z}_n^*$ is a *quadratic residue* modulo n if there is some s such that $x = s^2 \pmod{n}$.

Protocol QR

- **Recall** If n is an integer, then $x \in \mathbb{Z}_n^*$ is a *quadratic residue* modulo n if there is some s such that $x = s^2 \pmod{n}$.

  It is believed to be hard to tell whether x is a QR modulo n without knowing the factorization of n.

Protocol QR

- **Recall** If n is an integer, then $x \in \mathbb{Z}_n^*$ is a *quadratic residue* modulo n if there is some s such that $x = s^2 \pmod{n}$.

  It is believed to be hard to tell whether x is a QR modulo n without knowing the factorization of n.

  Some useful **facts**:

  $\diamond$ if n is prime, then $\mathbb{Z}_n^*$ has a generator g and x is a QR iff $x = g^i$ for an even i.

  $\diamond$ All the QRs form a *group*. If x is a QR, and y is a random QR, then xy is a random QR. For every $z \in QR_n$,

$$\Pr[xy = z] = 1/|QR_n|.$$

Protocol QR

- Statement P: x is a QR modulo n
  Public input: x, n; Prover – Alice; Verifier – Bob
  Prove's private input: w such that $x = w^2 \pmod{n}$

Protocol QR

- Statement P: x is a QR modulo n
  Public input: x, n; Prover – Alice; Verifier – Bob
  Prove's private input: w such that $x = w^2 \pmod{n}$

$P \rightarrow V$: Alice chooses random $u \leftarrow_R \mathbb{Z}_n^*$ and sends $y = u^2$ to Bob
$P \leftarrow V$: Bob chooses $b \leftarrow_R \{0, 1\}$
$P \rightarrow V$: If $b = 0$, Alice sends u to Bob. If $b = 1$, Alice sends $w \cdot u$.

Protocol QR

- Statement P: x is a QR modulo n
  Public input: x, n; Prover – Alice; Verifier – Bob
  Prove's private input: w such that $x = w^2 \pmod{n}$

$P \rightarrow V$: Alice chooses random $u \leftarrow_R \mathbb{Z}_n^*$ and sends $y = u^2$ to Bob
$P \leftarrow V$: Bob chooses $b \leftarrow_R \{0, 1\}$
$P \rightarrow V$: If $b = 0$, Alice sends u to Bob. If $b = 1$, Alice sends $w \cdot u$.

Verification: Let z denote the number sent by Alice.
Bob *accepts* the proof in the case $b = 0$, $z^2 = y \pmod{n}$, and in the case $b = 1$, $z^2 = xy \pmod{n}$.

Protocol QR

- Statement P: x is a QR modulo n
  Public input: x, n; Prover – Alice; Verifier – Bob
  Prove's private input: w such that $x = w^2 \pmod{n}$

  $P \rightarrow V$: Alice chooses random $u \leftarrow_R \mathbb{Z}_n^*$ and sends $y = u^2$ to Bob
  $P \leftarrow V$: Bob chooses $b \leftarrow_R \{0, 1\}$
  $P \rightarrow V$: If $b = 0$, Alice sends u to Bob. If $b = 1$, Alice sends $w \cdot u$.

  Verification: Let z denote the number sent by Alice.
  Bob *accepts* the proof in the case $b = 0$, $z^2 = y \pmod{n}$, and in the case $b = 1$, $z^2 = xy \pmod{n}$.

  We will analyze this protocol in *completeness, soundness, zero knowledge*.

Protocol QR – completeness

- *Completeness*: Whenever x is really a QR, Alice is given w such that $x = w^2 \pmod{n}$, and Alice and Bob follow the protocol, then Bob will *accept* the proof with probability 1.

Protocol QR – completeness

- *Completeness*: Whenever x is really a QR, Alice is given w such that $x = w^2 \pmod{n}$, and Alice and Bob follow the protocol, then Bob will *accept* the proof with probability 1.

  Soundness: If x is not a QR, then regardless of what Alice does, Bob will reject the proof with probability at least $1/2$.

Protocol QR – completeness

- *Completeness*: Whenever x is really a QR, Alice is given w such that $x = w^2 \pmod{n}$, and Alice and Bob follow the protocol, then Bob will *accept* the proof with probability 1.

  Soundness: If x is not a QR, then regardless of what Alice does, Bob will reject the proof with probability at least $1/2$.

  Alice may not follow the instructions in this protocol, and may possibly cheat. We model her strategy as a function P^*. We think of P^* as follows: on input the empty word, it gives a string y, and on input b, it gives a string z.

Protocol QR – soundness

- **Lemma 15.1** For every (possibly not efficiently computable) P^*, and (x, n) such that x is not a QR modulo n, we have

$$\Pr_{b \leftarrow \{0,1\}}[out_V \langle P^*, V_{x,b} \rangle = accept] \leq 1/2.$$

Protocol QR – soundness

- **Lemma 15.1** For every (possibly not efficiently computable) P^*, and (x, n) such that x is not a QR modulo n, we have

  $$\Pr_{b \leftarrow \{0,1\}}[out_V \langle P^*, V_{x,b} \rangle = accept] \leq 1/2.$$

  If two interactive algorithms A and B are running a protocol, we denote this execution by $\langle A, B \rangle$.

  $out_A \langle A, B \rangle$ – the output of A after this interaction is finished.
  $view_A \langle A, B \rangle$ – the *view* of A in the interaction: messages it received.

Protocol QR – soundness

- **Lemma 15.1** For every (possibly not efficiently computable) P^*, and (x, n) such that x is not a QR modulo n, we have

$$\Pr_{b \leftarrow \{0,1\}}[out_V \langle P^*, V_{x,b} \rangle = accept] \leq 1/2.$$

If two interactive algorithms A and B are running a protocol, we denote this execution by $\langle A, B \rangle$.

$out_A \langle A, B \rangle$ – the output of A after this interaction is finished.
$view_A \langle A, B \rangle$ – the *view* of A in the interaction: messages it received.

Proof. Suppose that $x \notin QR_n$. Denote by y the output of P^* on the empty input.

Protocol QR – soundness

- **Lemma 15.1** For every (possibly not efficiently computable) P^*, and (x, n) such that x is not a QR modulo n, we have

$$\Pr_{b \leftarrow \{0,1\}}[out_V \langle P^*, V_{x,b} \rangle = accept] \leq 1/2.$$

If two interactive algorithms A and B are running a protocol, we denote this execution by $\langle A, B \rangle$.

$out_A \langle A, B \rangle$ – the output of A after this interaction is finished.
$view_A \langle A, B \rangle$ – the *view* of A in the interaction: messages it received.

Proof. Suppose that $x \notin QR_n$. Denote by y the output of P^* on the empty input.

First, we cannot assume that y is a QR.

Second, y is the output before P^* sees the query b, and then y is independent of b.

Protocol QR – soundness

- **Lemma 15.1** For every (possibly not efficiently computable) P^*, and (x, n) such that x is not a QR modulo n, we have

$$\Pr_{b \leftarrow \{0,1\}}[out_V \langle P^*, V_{x,b} \rangle = accept] \leq 1/2.$$

Proof. Suppose that $x \notin QR_n$. Denote by y the output of P^* on the empty input.

First, we can not assume that y is a QR.

Second, y is the output before P^* sees the query b, and then y is independent of b.

Protocol QR – soundness

- **Lemma 15.1** For every (possibly not efficiently computable) P^*, and (x, n) such that x is not a QR modulo n, we have

$$\Pr_{b \leftarrow \{0,1\}}[out_V \langle P^*, V_{x,b} \rangle = accept] \leq 1/2.$$

Proof. Suppose that $x \notin QR_n$. Denote by y the output of P^* on the empty input.

First, we can not assume that y is a QR.

Second, y is the output before P^* sees the query b, and then y is independent of b.

Case 1: $y \in QR_n$. That is, $y = u^2 \pmod{n}$. With probability 1/2, Bob sends $b = 1$. Let $z = P^*(1)$. Bob will accept only if $z^2 = xy$. This is impossible since $z^2 y^{-1} = z^2 u^{-2} = xyy^{-1} = x$, but $x \notin QR_n$.

Protocol QR – soundness

- **Lemma 15.1** For every (possibly not efficiently computable) P^*, and (x, n) such that x is not a QR modulo n, we have

$$\Pr_{b\leftarrow\{0,1\}}[out_V\langle P^*, V_{x,b}\rangle = accept] \le 1/2.$$

Proof. Suppose that $x \notin QR_n$. Denote by y the output of P^* on the empty input.

First, we can not assume that y is a QR.

Second, y is the output before P^* sees the query b, and then y is independent of b.

Case 1: $y \in QR_n$. That is, $y = u^2 \pmod{n}$. With probability $1/2$, Bob sends $b = 1$. Let $z = P^*(1)$. Bob will accept only if $z^2 = xy$. This is impossible since $z^2y^{-1} = z^2u^{-2} = xyy^{-1} = x$, but $x \notin QR_n$.

Case 2: $y \notin QR_n$. With probability $1/2$, Bob sends $b = 0$. However, if $b = 0$, Alice has to come up with some z such that $z^2 = y$, impossible! Bob will also reject with probability $\ge 1/2$.

Protocol QR – zero knowledge

- We think of a possibly cheating verifier V^*. He can only sends either $b = 0$ or $b = 1$. Our goal is to show that: in both cases, he gets a random element in $\mathbb{Z}_n^*$, leaking no info about the QR of x.

Protocol QR – zero knowledge

- We think of a possibly cheating verifier V^*. He can only sends either $b = 0$ or $b = 1$. Our goal is to show that: in both cases, he gets a random element in $\mathbb{Z}_n^*$, leaking no info about the QR of x.

  Idea: A proof is *zero knowledge* if whatever Bob learns, he could have learned by himself without any interaction with Alice.

Protocol QR – zero knowledge

- We think of a possibly cheating verifier V^*. He can only sends either $b = 0$ or $b = 1$. Our goal is to show that: in both cases, he gets a random element in $\mathbb{Z}_n^*$, leaking no info about the QR of x.

  Idea: A proof is *zero knowledge* if whatever Bob learns, he could have learned by himself without any interaction with Alice.

An Example

- What does Bob see?
  - randomly-generated keys
  - randomly-generated colors

  Because Bob could have generated those keys and colors by himself, he learns nothing from the graph coloring.

Protocol QR – zero knowledge

- **Definition 15.2** A prove strategy P is (T, ϵ)-zero knowledge if for every T-time cheating strategy V^* there exists a $poly(T)$-time non-interactive algorithm S (called the *simulator* for V^*) such that for every valid public input x and private input w, the following two random variables are (T, ϵ)-computationally indistinguishable:
  - $view_{V^*}\langle P_{U_m,x,w}, V^*\rangle$, where m is the number of random coins P uses.
  - $S(x)$. Note that S can be probabilistic and so this is a r.v.

Protocol QR – zero knowledge

- **Definition 15.2** A prove strategy P is (T, ϵ)-zero knowledge if for every T-time cheating strategy V^* there exists a $poly(T)$-time non-interactive algorithm S (called the *simulator* for V^*) such that for every valid public input x and private input w, the following two random variables are (T, ϵ)-computationally indistinguishable:
  - $view_{V^*}\langle P_{U_m,x,w}, V^*\rangle$, where m is the number of random coins P uses.
  - $S(x)$. Note that S can be probabilistic and so this is a r.v.

The *simulator* S only gets the public input and has no interaction with P, but still manages to output something indistinguishable from whatever V^* learned in the interaction.

Protocol QR – zero knowledge

- **Lemma 15.3** The prover of Protocol QR is $(\infty, 2^{-|x|})$ - zero knowledge.

Protocol QR – zero knowledge

- **Lemma 15.3** The prover of Protocol QR is $(\infty, 2^{-|x|})$ - zero knowledge.

  Proof. Let V^* be a possibly cheating verifier. The simulator S will do the following (S can depend on V^*):

  1. **Input**: x, n such that $x \in QR_n$.
  2. Choose $b' \leftarrow_R \{0, 1\}$.
  3. Choose $z \leftarrow_R QR_n$.
  4. If $b' = 0$, compute $y = z^2$. Otherwise $(b' = 1)$, compute $y = z^2 x^{-1}$.
  5. Invoke V^* on the message y to obtain a bit b.
  6. If $b = b'$, then output $\langle y, z \rangle$. Otherwise, go back to Step 2.

Protocol QR – zero knowledge

- **Lemma 15.3** The prover of Protocol QR is $(\infty, 2^{-|x|})$ - zero knowledge.

  Proof. Let V^* be a possibly cheating verifier. The simulator S will do the following (S can depend on V^*):

  1. **Input**: x, n such that $x \in QR_n$.
  2. Choose $b' \leftarrow_R \{0, 1\}$.
  3. Choose $z \leftarrow_R QR_n$.
  4. If $b' = 0$, compute $y = z^2$. Otherwise ($b' = 1$), compute $y = z^2 x^{-1}$.
  5. Invoke V^* on the message y to obtain a bit b.
  6. If $b = b'$, then output $\langle y, z \rangle$. Otherwise, go back to Step 2.

  We do not even know whether this algorithm loops forever or not.

Protocol QR – zero knowledge

- **Lemma 15.4** In both case $b' = 0$ and $b' = 1$, the message y is a random element in QR_n.

Protocol QR – zero knowledge

- **Lemma 15.4** In both case $b' = 0$ and $b' = 1$, the message y is a random element in QR_n.

  Proof. In the case $b' = 0$, this is obvious.

Protocol QR – zero knowledge

- **Lemma 15.4** In both case $b' = 0$ and $b' = 1$, the message y is a random element in QR_n.

  Proof. In the case $b' = 0$, this is obvious.

  In the case $b' = 1$, $y = x^{-1}z^2$, where z^2 is a random element in QR_n. So is y since $x \in QR_n$.

Protocol QR – zero knowledge

- **Lemma 15.4** In both case $b' = 0$ and $b' = 1$, the message y is a random element in QR_n.

  Proof. In the case $b' = 0$, this is obvious.

  In the case $b' = 1$, $y = x^{-1}z^2$, where z^2 is a random element in QR_n. So is y since $x \in QR_n$.

  This implies that y is independent of b'. We have already known that $b = V^*(y)$ is also independent of b' and hence we have $\Pr[b = b'] = 1/2$. If we run the algorithm for k steps, we will halt with very high probability $(1 - 2^{-k})$.

Protocol QR – zero knowledge

- **Lemma 15.4** In both case $b' = 0$ and $b' = 1$, the message y is a random element in QR_n.

  Proof. In the case $b' = 0$, this is obvious.

  In the case $b' = 1$, $y = x^{-1}z^2$, where z^2 is a random element in QR_n. So is y since $x \in QR_n$.

  This implies that y is independent of b'. We have already known that $b = V^*(y)$ is also independent of b' and hence we have $\Pr[b = b'] = 1/2$. If we run the algorithm for k steps, we will halt with very high probability $(1 - 2^{-k})$.

  Lemma 15.5 The output of the simulator S is distributed identically to the view of V^* in an interaction with an honest prover.

Protocol QR – zero knowledge

- **Lemma 15.4** In both case $b' = 0$ and $b' = 1$, the message y is a random element in QR_n.

  Proof. In the case $b' = 0$, this is obvious.

  In the case $b' = 1$, $y = x^{-1}z^2$, where z^2 is a random element in QR_n. So is y since $x \in QR_n$.

  This implies that y is independent of b'. We have already known that $b = V^*(y)$ is also independent of b' and hence we have $\Pr[b = b'] = 1/2$. If we run the algorithm for k steps, we will halt with very high probability $(1 - 2^{-k})$.

  Lemma 15.5 The output of the simulator S is distributed identically to the view of V^* in an interaction with an honest prover.

  Proof. For both the prover and the simulator, if $b = 0$, then z is a random root of y; if $b = 1$, then z is a random root of xy.

Schnorr's identification protocol

- Statement P: Alice knows DL of h, w.r.t. g, these are in group $G = \mathbb{Z}_p$.

  Public input: g, h; Prover – Alice; Verifier – Bob
  Prover's private input: x such that $h = g^x$

  $P \rightarrow V$: Alice chooses random $r \leftarrow_R \mathbb{Z}_p$ and sends $a = g^r$ to Bob
  $P \leftarrow V$: Bob chooses $b \leftarrow_R \mathbb{Z}_p$ and sends b to Alice
  $P \rightarrow V$: Alice sends $c = r + xb \pmod{p}$ to Bob.

  Verification: Bob verifies that $ah^b = g^c$.

Schnorr's identification protocol

- *Completeness*: obvious

Schnorr's identification protocol

- *Completeness*: obvious

  Proof of knowledge: if $b \neq b'$ then given a and $b \neq b'$ and $c \neq c'$ such that $ah^b = g^c$ and $ah^{b'} = g^{c'}$, we get $h^{b-b'} = g^{c-c'}$. Since we know b and b', we can take this to the power $(b-b')^{-1} \pmod{p}$ to get an equation $h = g^x$.

Schnorr's identification protocol

- *Completeness*: obvious

Proof of knowledge: if $b \neq b'$ then given a and $b \neq b'$ and $c \neq c'$ such that $ah^b = g^c$ and $ah^{b'} = g^{c'}$, we get $h^{b-b'} = g^{c-c'}$. Since we know b and b', we can take this to the power $(b-b')^{-1} \pmod{p}$ to get an equation $h = g^x$.

Honest verifier zero knowledge: The simulator S does the following: choose $b, c \leftarrow_R \mathbb{Z}_p$, choose a as $h^{-b}g^c$.

Homomorphic encryption

- **Definition 15.6** (*Group homomorphism*)
  Two groups G and G' are *homomorphic* if there exists a function (*homomorphism*) $f : G \to G'$ such that for all $x, y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$.

Homomorphic encryption

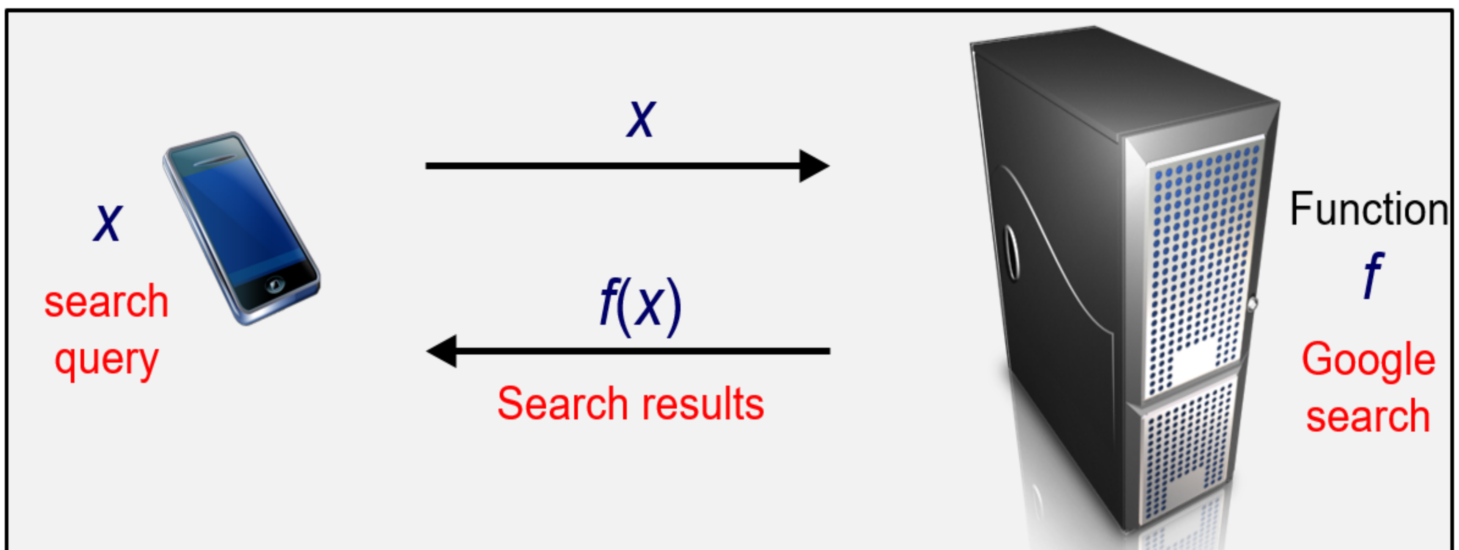
- **Definition 15.6** (*Group homomorphism*)
  Two groups G and G' are *homomorphic* if there exists a function (*homomorphism*) $f : G \to G'$ such that for all $x, y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$.

  Why do we need *homomorphic encryption*?

Homomorphic encryption

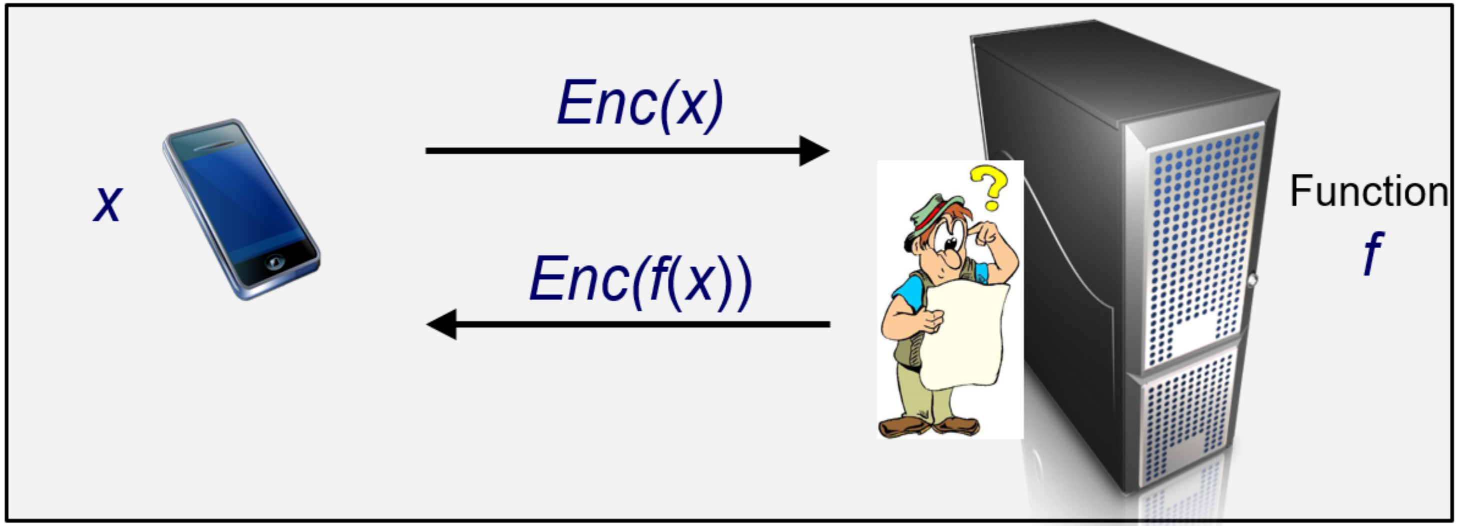
- **Definition 15.6** (*Group homomorphism*)
  Two groups G and G' are *homomorphic* if there exists a function (*homomorphism*) $f : G \to G'$ such that for all $x, y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$.

  Why do we need *homomorphic encryption*?



- **Definition 15.6** (*Group homomorphism*)
  Two groups G and G' are *homomorphic* if there exists a function (*homomorphism*) $f : G \to G'$ such that for all $x, y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$.

  Why do we need *homomorphic encryption*?



Computing on encrypted data

- Recall RSA encryption

  $E(m_1) = m_1^e \bmod n$, $E(m_2) = m_2^e \bmod n$

Computing on encrypted data

- Recall RSA encryption

$$E(m_1) = m_1^e \bmod n,\ E(m_2) = m_2^e \bmod n$$

$$E(m_1) \cdot E(m_2) = m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e = E(m_1 \cdot m_2)$$

Computing on encrypted data

- Recall RSA encryption

  $E(m_1) = m_1^e \bmod n,\ E(m_2) = m_2^e \bmod n$

  $E(m_1) \cdot E(m_2) = m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e = E(m_1 \cdot m_2)$

  RSA is multiplicatively homomorphic, but not additively homomorphic.

Computing on encrypted data

- Recall RSA encryption

  $E(m_1) = m_1^e \bmod n,\ E(m_2) = m_2^e \bmod n$

  $E(m_1) \cdot E(m_2) = m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e = E(m_1 \cdot m_2)$

  RSA is multiplicatively homomorphic, but not additively homomorphic.

  We need **both**!

Computing on encrypted data

- Recall RSA encryption

  $E(m_1) = m_1^e \bmod n$, $E(m_2) = m_2^e \bmod n$

  $E(m_1) \cdot E(m_2) = m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e = E(m_1 \cdot m_2)$

  RSA is multiplicatively homomorphic, but not additively homomorphic.

  We need **both**!

  What people really wanted was the ability to do arbitrary computing on encrypted data, and this requires the abibility to compute both sums and products.

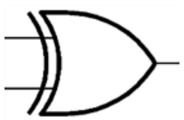
Computing on encrypted data

- Why SUMs and PRODUCTs?

Computing on encrypted data

- Why SUMs and PRODUCTs?

SUM



XOR

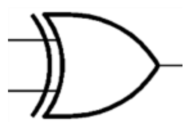
PRODUCT

AND

Computing on encrypted data

- Why SUMs and PRODUCTs?

SUM



XOR

$x + y \bmod 2$

PRODUCT

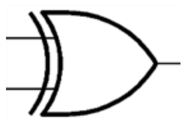
AND

$x \cdot y \bmod 2$

Computing on encrypted data

- Why SUMs and PRODUCTs?

SUM



XOR

$x + y \bmod 2$

PRODUCT

AND

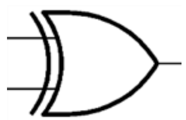
$x \cdot y \bmod 2$

{XOR, AND} is complete, i.e.,
any function is a combination of XOR and AND. (e.g., OR)

Computing on encrypted data

- Why SUMs and PRODUCTs?

SUM



XOR

$x + y \bmod 2$

PRODUCT

AND

$x \cdot y \bmod 2$

{XOR, AND} is complete, i.e.,
any function is a combination of XOR and AND. (e.g., OR)

Example
$x \ OR \ y = x + y + x \cdot y \bmod 2.$

Computing on encrypted data

- Because $\{$XOR, AND$\}$ is *complete*, if we can compute SUMs and PRODUCTs on encrypted bits, we can compute any function on encrypted inputs.

Computing on encrypted data

- Because $\{$XOR, AND$\}$ is *complete*, if we can compute SUMs and PRODUCTs on encrypted bits, we can compute any function on encrypted inputs.

Fully-homomorphic encryption!

We can delegate arbitrary processing of data without giving away access to it.

Computing on encrypted data

- Because {XOR, AND} is *complete*, if we can compute SUMs and PRODUCTs on encrypted bits, we can compute any function on encrypted inputs.

  Fully-homomorphic encryption!

  We can delegate arbitrary processing of data without giving away access to it.

  Applications: *private cloud computing, private information retrieval, multi-party secure computation, encrypted search, ...*

Fully homomorphic encryption

Fully Homomorphic Encryption Using Ideal Lattices

Craig Gentry
Stanford University and IBM Watson
cgentry@cs.stanford.edu

ABSTRACT

We propose a fully homomorphic encryption scheme – i.e., a scheme that allows one to evaluate circuits over encrypted data without being able to decrypt. Our solution comes in three steps. First, we provide a general result – that, to construct an encryption scheme that permits evaluation of *arbitrary circuits*, it suffices to construct an encryption

duced by Rivest, Adleman and Dertouzos [54] shortly after the invention of RSA by Rivest, Adleman and Shamir [55]. Basic RSA is a multiplicatively homomorphic encryption scheme – i.e., given RSA public key pk $= (N, e)$ and ciphertexts $\{\psi_i \leftarrow \pi_i^e \bmod N\}$, one can efficiently compute $\prod_i \psi_i = (\prod_i \pi_i)^e \bmod N$, a ciphertext that encrypts the product of the original plaintexts. Rivest et al. [54] asked

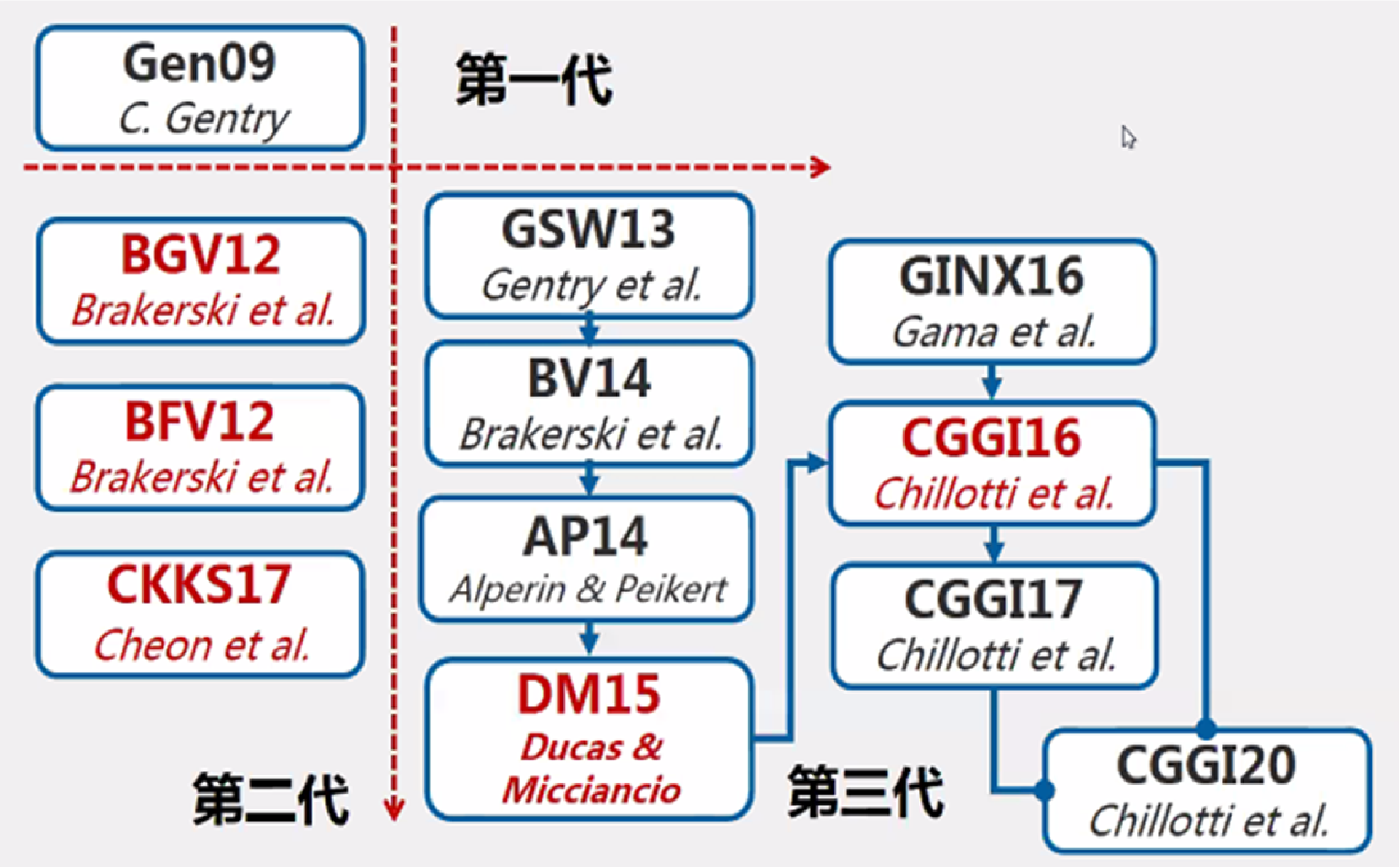
Fully Homomorphic Encryption over the Integers

Marten van Dijk[1], Craig Gentry[2], Shai Halevi[2], and Vinod Vaikuntanathan[2]

[1] MIT CSAIL
[2] IBM Research

Abstract. We construct a simple fully homomorphic encryption scheme, using only elementary modular arithmetic. We use Gentry's technique to construct a fully homomorphic scheme from a "bootstrappable" somewhat homomorphic scheme. However, instead of using ideal lattices over a

Fully homomorphic encryption



Fully homomorphic encryption

Library	Developed by	FHE Scheme
HElib	IBM	BGV/CKKS
Microsoft SEAL	Microsoft	BFV/CKKS
PALISADE	MIT, UCSD etc.	BFV/BGV etc.
HEAAN	Seoul National University	CKKS

- **Definition 15.7** (*Fully homomorphic encryption*)
  We say that a CPA-secure public key encryption scheme (G, E, D) with one bit messages is *fully homomorphic* if there exists an algorithm *NAND* such that for every $(e, d) \leftarrow G(1^n)$, $a, b \in \{0, 1\}$, and $\hat{a} \leftarrow E_e(a)$, $\hat{b} \leftarrow E_e(b)$,

$$NAND_e(\hat{a}, \hat{b}) \approx E_e(a \ NAND \ b),$$

  where $a \ NAND \ b = \neg(a \wedge b)$.

Gentry's scheme

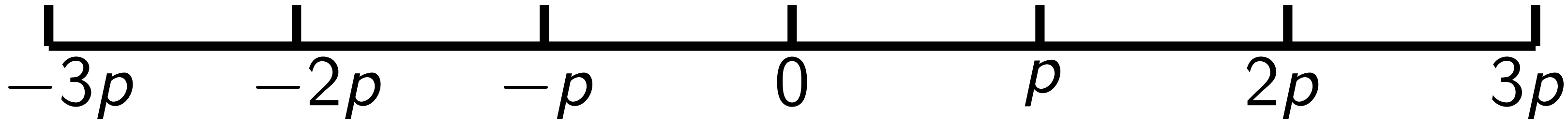
- Secret key: large odd number p

Gentry's scheme

- Secret key: large odd number p

  To encrypt a bit b:

  – pick a random "large" multiple of p, say, $q \cdot p$.



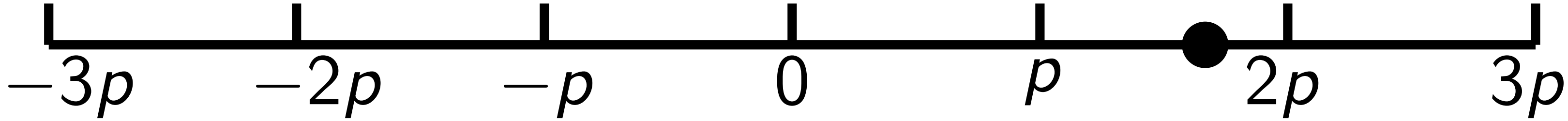
Gentry's scheme

- Secret key: large odd number p

  To encrypt a bit b:

  – pick a random "large" multiple of p, say, $q \cdot p$.

  – pick a random "small" number $2 \cdot r + b$ (this is even if $b = 0$, and odd if $b = 1$).



Gentry's scheme

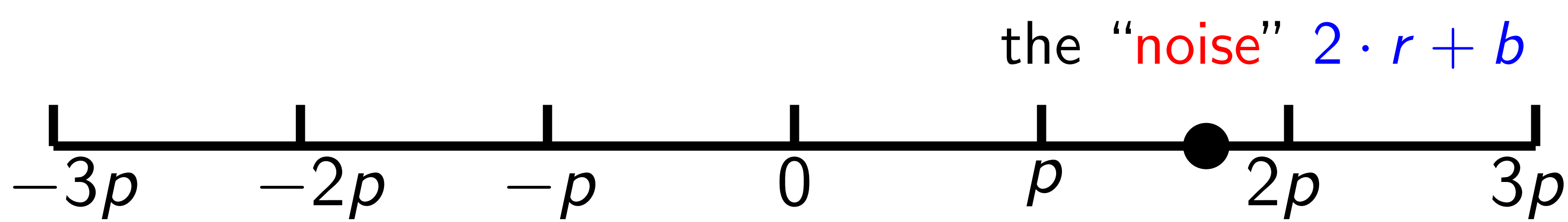
- Secret key: large odd number p

  To encrypt a bit b:

  – pick a random "large" multiple of p, say, $q \cdot p$.

  – pick a random "small" number $2 \cdot r + b$ (this is even if $b = 0$, and odd if $b = 1$).

  – ciphertext $c = q \cdot p + 2 \cdot r + b$



Gentry's scheme

- Secret key: large odd number p

To encrypt a bit b:

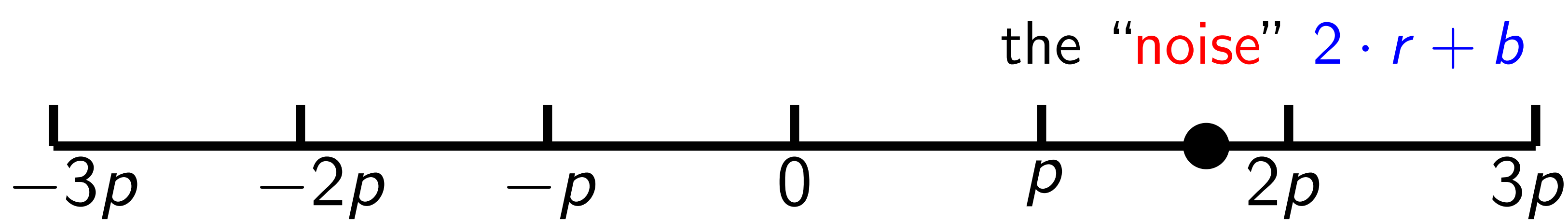
– pick a random "large" multiple of p, say, $q \cdot p$.

– pick a random "small" number $2 \cdot r + b$ (this is even if $b = 0$, and odd if $b = 1$).

– ciphertext $c = q \cdot p + 2 \cdot r + b$

To decrypt a ciphertext c:

– Taking c mod p recovers the noise $2 \cdot r + b$.



Gentry's Scheme

- How secure is this scheme?

  If there is no "noise" ($r = 0$), given two encryptions of 0 ($q_1 p$ and $q_2 p$), then we can recover the secret key $p = \gcd(q_1 p, q_2 p)$.

Gentry's Scheme

- How secure is this scheme?

  If there is no "noise" ($r = 0$), given two encryptions of 0 ($q_1 p$ and $q_2 p$), then we can recover the secret key $p = \gcd(q_1 p, q_2 p)$.

  If there is "noise", the gcd attack does not work, and it is believed that there is no other attack.

Gentry's Scheme

- How secure is this scheme?

  If there is no "noise" ($r = 0$), given two encryptions of 0 ($q_1 p$ and $q_2 p$), then we can recover the secret key $p = \gcd(q_1 p, q_2 p)$.

  If there is "noise", the gcd attack does not work, and it is believed that there is no other attack.

  (the *approximate* gcd *assumption*)

Gentry's Scheme: XOR

- XORing two encrypted bits:
  - $c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$
  - $c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$

Gentry's Scheme: XOR

- XORing two encrypted bits:
  - $c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$
  - $c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$

$$c_1 + c_2 = (q_1 + q_2) \cdot p + 2 \cdot (r_1 + r_2) + (b_1 + b_2)$$

Gentry's Scheme: XOR

- XORing two encrypted bits:
  - $c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$
  - $c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$

$$c_1 + c_2 = (q_1 + q_2) \cdot p + \boxed{2 \cdot (r_1 + r_2) + (b_1 + b_2)}$$

ODD if $b_1 = 0, b_2 = 1$
or $b_1 = 1, b_2 = 0$

EVEN if $b_1 = 0, b_2 = 0$
or $b_1 = 1, b_2 = 1$

Gentry's Scheme: XOR

- XORing two encrypted bits:
  - $c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$
  - $c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$

$$c_1 + c_2 = (q_1 + q_2) \cdot p + \boxed{2 \cdot (r_1 + r_2) + (b_1 + b_2)}$$

ODD if $b_1 = 0, b_2 = 1$
or $b_1 = 1, b_2 = 0$

EVEN if $b_1 = 0, b_2 = 0$
or $b_1 = 1, b_2 = 1$

$$c_1 + c_2 = Enc(b_1 + b_2)$$

Gentry's Scheme: AND

- XORing two encrypted bits:
  - $c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$
  - $c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$

$$c_1 \cdot c_2 = (c_2 q_1 + c_1 q_2 - q_1 q_2)p + 2(2 r_1 r_2 + r_1 b_2 + r_2 b_1) + (b_1 b_2)$$

Gentry's Scheme: AND

- XORing two encrypted bits:
  - $c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$
  - $c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$

$$c_1 \cdot c_2 = (c_2 q_1 + c_1 q_2 - q_1 q_2)p + \boxed{2(2 r_1 r_2 + r_1 b_2 + r_2 b_1) + (b_1 b_2)}$$

ODD if $b_1 = 1, b_2 = 1$

EVEN if $b_1 = 0, b_2 = 0$
or $b_1 = 0, b_2 = 1$
or $b_1 = 1, b_2 = 0$

Gentry's Scheme: AND

- XORing two encrypted bits:
  - $c_1 = q_1 \cdot p + (2 \cdot r_1 + b_1)$
  - $c_2 = q_2 \cdot p + (2 \cdot r_2 + b_2)$

$$c_1 \cdot c_2 = (c_2 q_1 + c_1 q_2 - q_1 q_2)p + \boxed{2(2r_1 r_2 + r_1 b_2 + r_2 b_1) + (b_1 b_2)}$$

ODD if $b_1 = 1, b_2 = 1$

EVEN if $b_1 = 0, b_2 = 0$
or $b_1 = 0, b_2 = 1$
or $b_1 = 1, b_2 = 0$

$$c_1 \cdot c_2 = Enc(b_1 \cdot b_2)$$

Problem: the noise grows!!!

- $c_1 + c_2 = (q_1 + q_2) \cdot p + 2 \cdot (r_1 + r_2) + (b_1 + b_2)$

Problem: the noise grows!!!

- $c_1 + c_2 = (q_1 + q_2) \cdot p + 2 \cdot (r_1 + r_2) + (b_1 + b_2)$

  "noise" $= 2\cdot$ (initial noise)

Problem: the noise grows!!!

- $c_1 + c_2 = (q_1 + q_2) \cdot p + 2 \cdot (r_1 + r_2) + (b_1 + b_2)$

  "noise" $=$ 2· (initial noise)

  $c_1 \cdot c_2 = (c_2 q_1 + c_1 q_2 - q_1 q_2) p + 2(2 r_1 r_2 + r_1 b_2 + r_2 b_1) + (b_1 b_2)$

  "noise" $=$ (initial noise)^2

Problem: the noise grows!!!

- $c_1 + c_2 = (q_1 + q_2) \cdot p + 2 \cdot (r_1 + r_2) + (b_1 + b_2)$

  $$\text{“noise”} = 2 \cdot \text{(initial noise)}$$

  $c_1 \cdot c_2 = (c_2 q_1 + c_1 q_2 - q_1 q_2) p + 2(2 r_1 r_2 + r_1 b_2 + r_2 b_1) + (b_1 b_2)$

  $$\text{“noise”} = \text{(initial noise)}^2$$

  What is the problem?

Problem: the noise grows!!!

- $c_1 + c_2 = (q_1 + q_2) \cdot p + 2 \cdot (r_1 + r_2) + (b_1 + b_2)$

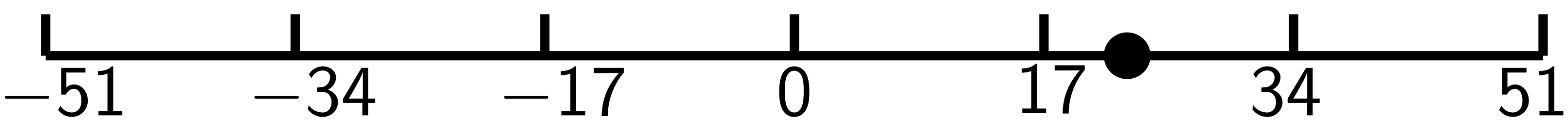
$$\text{“noise”} \;=\; 2\cdot\ (\text{initial noise})$$

$$c_1 \cdot c_2 = (c_2 q_1 + c_1 q_2 - q_1 q_2)p + 2(2r_1 r_2 + r_1 b_2 + r_2 b_1) + (b_1 b_2)$$

$$\text{“noise”} \;=\; (\text{initial noise})^2$$

What is the problem?

the “noise” $= 14$, decryption will recover noise $= 3$



Problem: the noise grows!!!

- $c_1 + c_2 = (q_1 + q_2) \cdot p + 2 \cdot (r_1 + r_2) + (b_1 + b_2)$

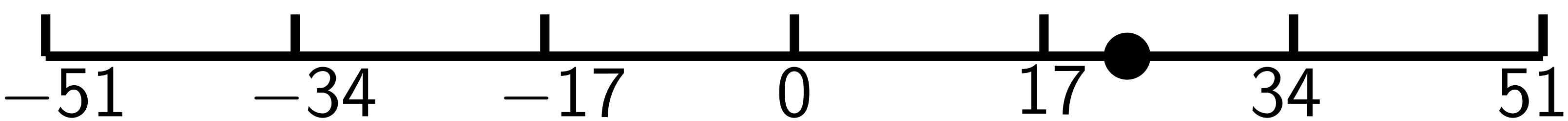
  "noise" $= 2\cdot$ (initial noise)

  $c_1 \cdot c_2 = (c_2 q_1 + c_1 q_2 - q_1 q_2)p + 2(2r_1 r_2 + r_1 b_2 + r_2 b_1) + (b_1 b_2)$

  "noise" $=$ (initial noise)^2

  What is the problem?

  the "noise" $= 14$, decryption will recover noise $= 3$



  If the $|noise| > p/2$, then decryption will be incorrect!

The "Bootstrapping Method"

- Problem: Add and Mult increase noise (Add doubles, and Mult squares the noise). So, we want to do noise-reduction.

The "Bootstrapping Method"

- Problem: Add and Mult increase noise (Add doubles, and Mult squares the noise). So, we want to do noise-reduction.

  $\mathcal{Q}$: What is the best noise-reduction procedure?

The "Bootstrapping Method"

- Problem: Add and Mult increase noise (Add doubles, and Mult squares the noise). So, we want to do noise-reduction.

  $\mathcal{Q}$: What is the best noise-reduction procedure?

  $\mathcal{A}$: something that kills all noise and recovers the message...

The “Bootstrapping Method”

- Problem: Add and Mult increase noise (Add doubles, and Mult squares the noise). So, we want to do noise-reduction.

  $\mathcal{Q}$: What is the best noise-reduction procedure?

  $\mathcal{A}$: something that kills all noise and recovers the message...

  Decryption!

The "Bootstrapping Method"

- Problem: Add and Mult increase noise (Add doubles, and Mult squares the noise). So, we want to do noise-reduction.

  $\mathcal{Q}$: What is the best noise-reduction procedure?

  $\mathcal{A}$: something that kills all noise and recovers the message...

  Decryption!

  $\mathcal{Q}'$: But we cannot release the secret key! Then how?

The "Bootstrapping Method"

- Problem: Add and Mult increase noise (Add doubles, and Mult squares the noise). So, we want to do noise-reduction.

  $\mathcal{Q}$: What is the best noise-reduction procedure?

  $\mathcal{A}$: something that kills all noise and recovers the message...

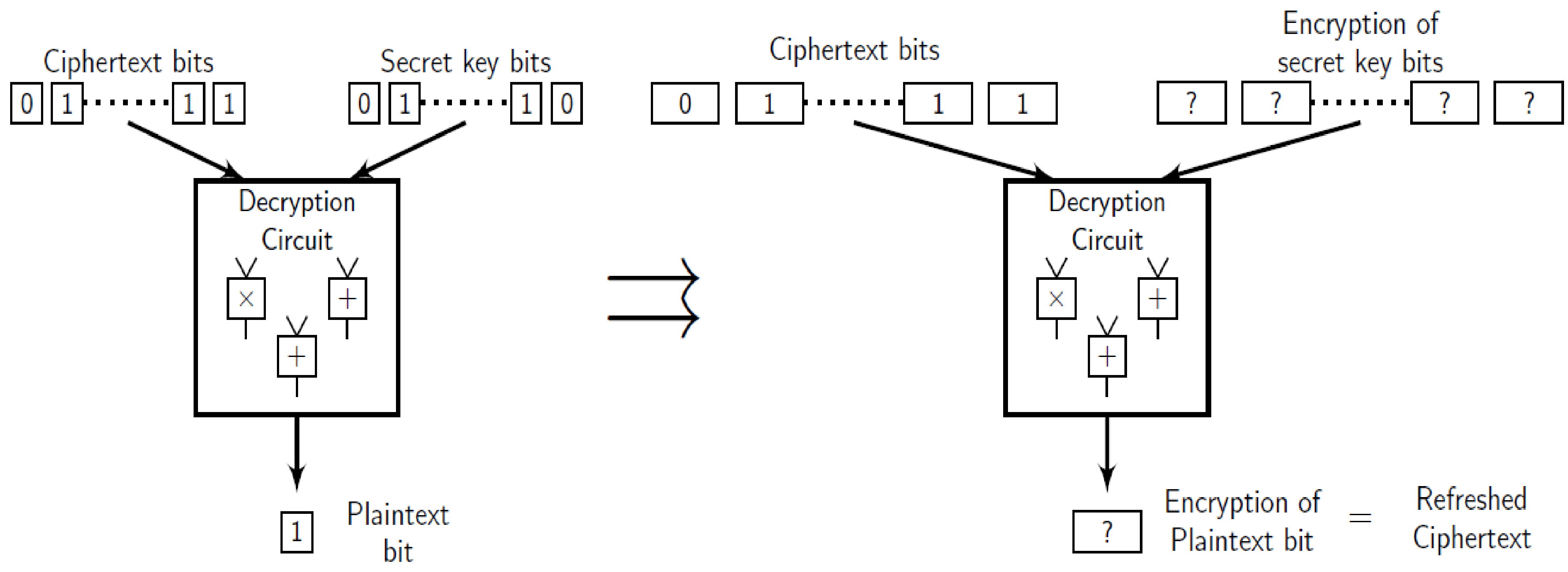
  ## Decryption!

  $\mathcal{Q}'$: But we cannot release the secret key! Then how?

  $\mathcal{A}'$: release $Enc(secret\ key)$

  Instead of recovering the bit plaintext b, one gets an encryption of this bit plaintext, i.e., yet another ciphertext for the same plaintext.

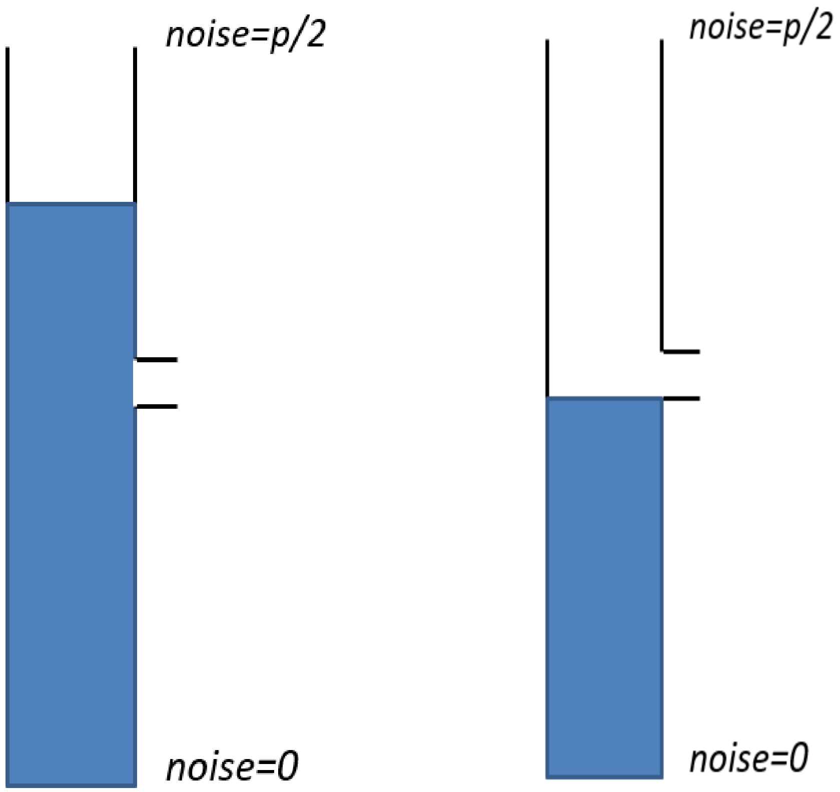
The "Bootstrapping Method"



Observation: The original $Enc(b)$ and the "refreshed" $Enc(b)$ have different noise levels. Regardless of the noise in the original $Enc(b)$, the noise level in the "refreshed" $Enc(b)$ is FIXED.

The "Bootstrapping Method"

- **Bottomline**: whenever noise level increases beyond a limit, use bootstrapping to reset it to a fixed level.



More Readings

2011 IEEE 52nd Annual Symposium on Foundations of Computer Science

Computing Blindfolded: New Developments in Fully Homomorphic Encryption

Vinod Vaikuntanathan
University of Toronto

Abstract— A fully homomorphic encryption scheme enables computation of arbitrary functions on encrypted data. Fully homomorphic encryption has long been regarded as cryptography's prized "holy grail" – extremely useful yet rather elusive. Starting with the groundbreaking work of Gentry in 2009, the last three years have witnessed numerous constructions of fully homomorphic encryption involving novel mathematical techniques, and a number of exciting applications. We will take the reader through a journey of these developments and provide a glimpse of the exciting research directions that lie ahead.

arbitrary computations to the "cloud" and the ability to store all data encrypted and perform computations on encrypted data, decrypting only when necessary.

Fully Homomorphic encryption is a special type of encryption system that permits *arbitrarily complex computation* on encrypted data. Long regarded as a "holy grail" of cryptography, fully homomorphic encryption was first shown to be possible in the recent, breakthrough work of Gentry. We will take the reader through

Good Luck!