CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

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

Digital signatures

- Provide *integrity* in the public-key setting
- Analogous to *message authentication codes*, but some key differences

Digital signatures

- Provide *integrity* in the public-key setting

- Analogous to *message authentication codes*, but some key differences

	Private Key	Public Key
Secrecy	private key encryption	public key encryption
Integrity	MAC	??

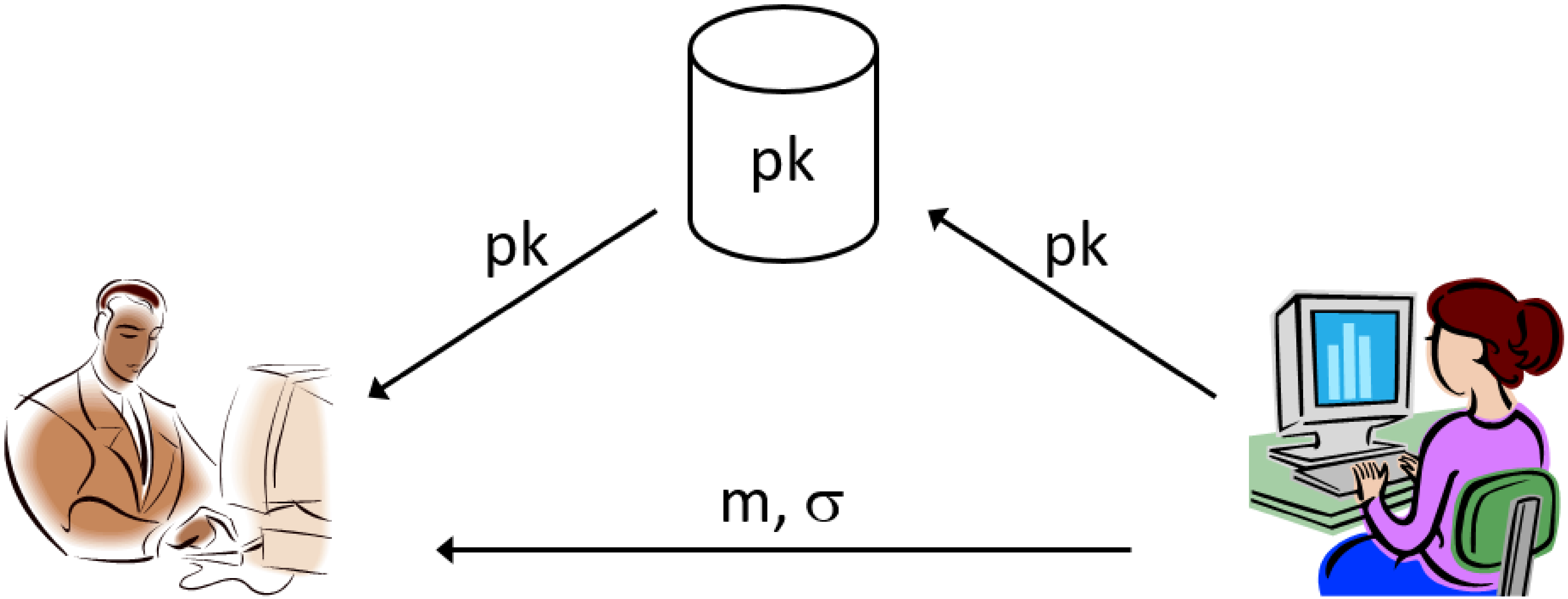
Digital signatures

- A *signature scheme* is defined by three PPT algorithms (*Gen*, *Sign*, *Vrfy*):
  - *Gen*: takes as input 1^n; outputs pk, sk
  - *Sign*: takes as input a private key sk and a message $m \in \{0,1\}^*$; outputs *signature* σ: $\sigma \leftarrow Sign_{sk}(m)$
  - *Vrfy*: takes public key pk, message m, and signature σ as input; outputs 1 or 0

For all m and all pk, sk output by *Gen*,

$$Vrfy_{pk}(m, Sign_{sk}(m)) = 1$$

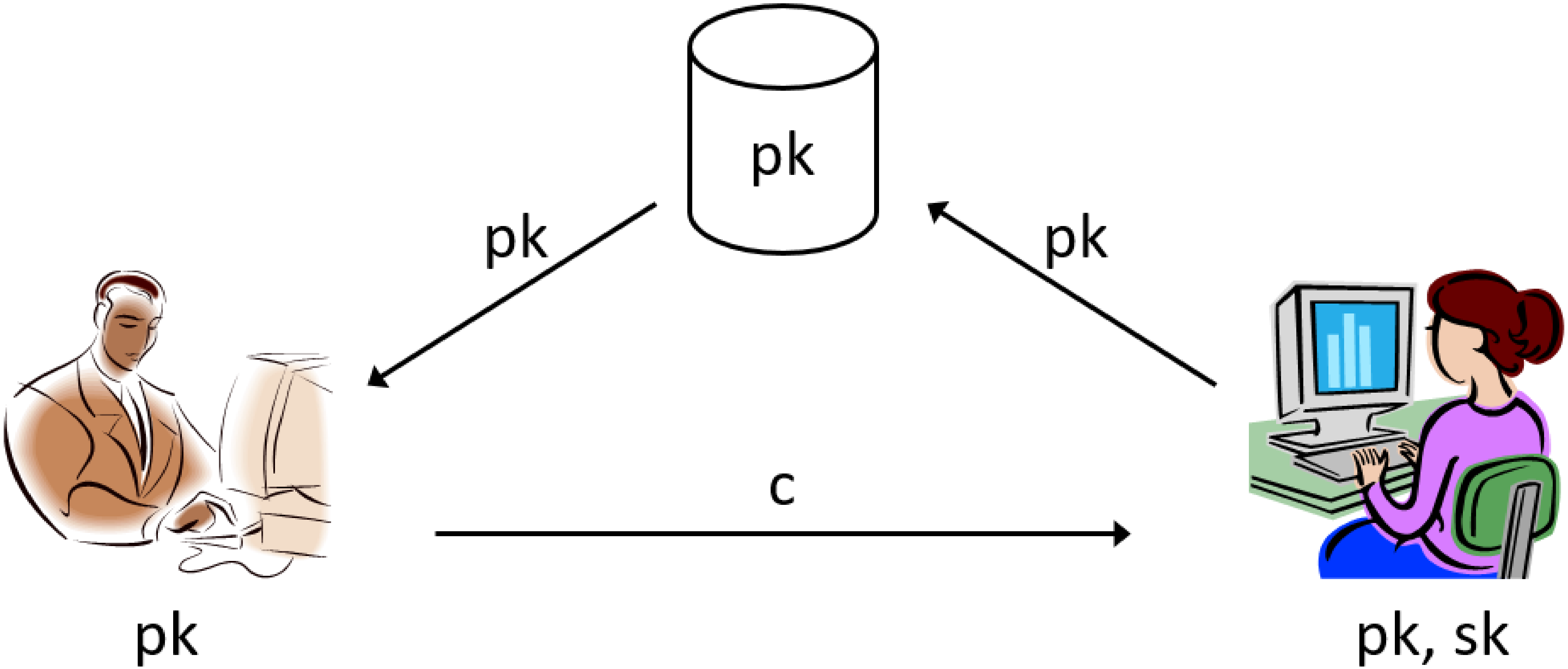
Digital signatures



pk $\qquad\qquad\qquad\qquad$ pk, sk

$1 \stackrel{?}{=} \mathrm{Vrfy}_{pk}(m, \sigma)$ $\qquad\qquad$ $\sigma = \mathrm{Sign}_{sk}(m)$

Public-key encryption



$c \leftarrow \mathrm{Enc}_{pk}(m)$

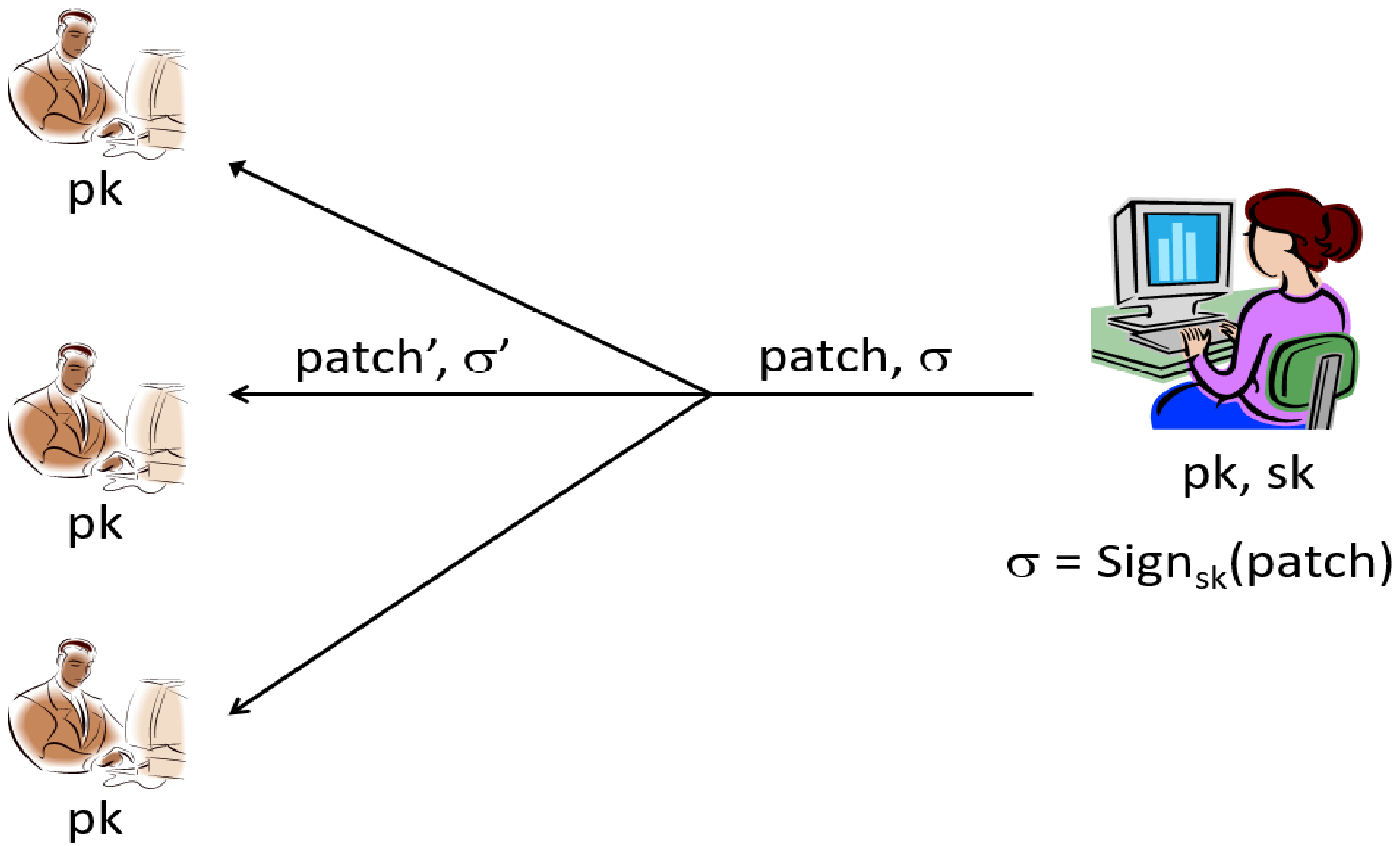
$m = \mathrm{Dec}_{sk}(c)$

Security (informal)

- Even after observing signatures on multiple messages, an attacker should be unable to *forge* a valid signature on a new message

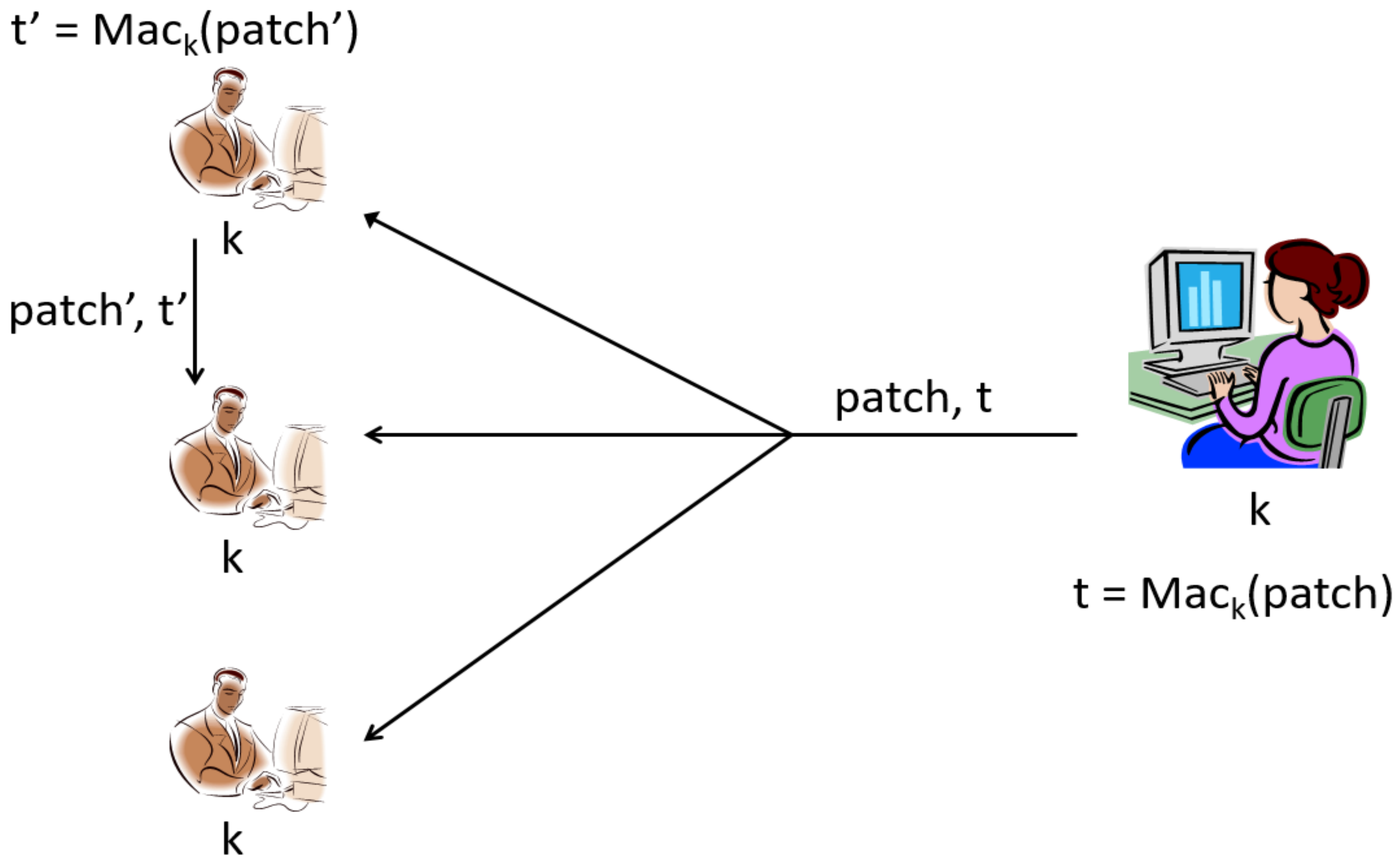
Security (informal)

- Even after observing signatures on multiple messages, an attacker should be unable to *forge* a valid signature on a new message
- Prototypical application



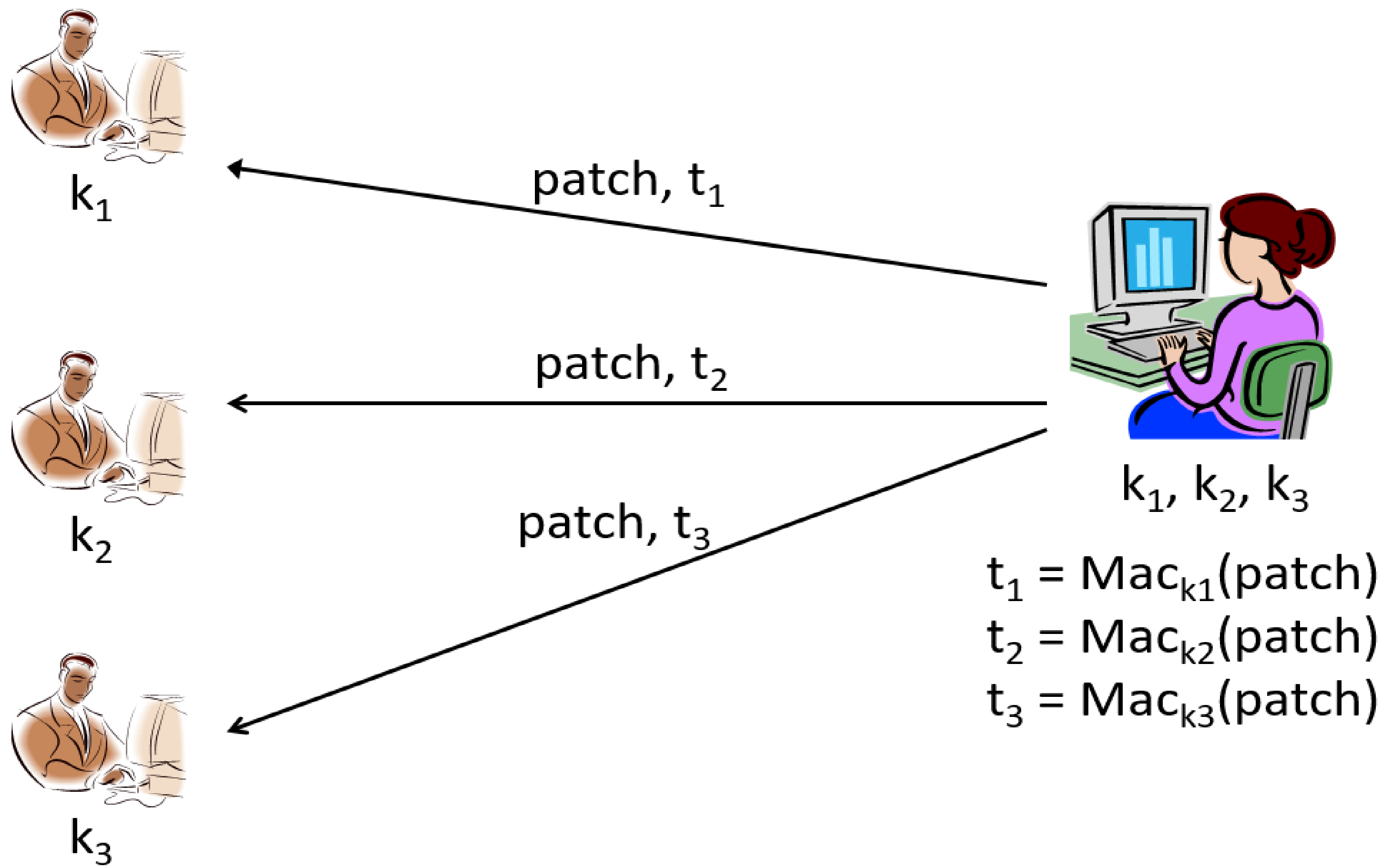
Security (informal)

- Even after observing signatures on multiple messages, an attacker should be unable to *forge* a valid signature on a new message
- Comparison to MACs?



Security (informal)

- Even after observing signatures on multiple messages, an attacker should be unable to *forge* a valid signature on a new message
- Comparison to MACs?



Comparison to MACs

- *Public verifiability*
  - "Anyone" can verify a signature
  - Only a holder of the key can verify a MAC tag

Comparison to MACs

- *Public verifiability*
  - "Anyone" can verify a signature
  - Only a holder of the key can verify a MAC tag

$\Rightarrow$ *Transferability*

  - Can forward a signature to someone else

Comparison to MACs

- *Public verifiability*
  - "Anyone" can verify a signature
  - Only a holder of the key can verify a MAC tag

$\Rightarrow$ *Transferability*

  - Can forward a signature to someone else

$\Rightarrow$ *Non-repudiation*

Non-repudiation

- Signer cannot (easily) deny issuing a signature
  - Crucial for legal applications
  - Judge can verify signature using a public copy of *pk*

Non-repudiation

- Signer cannot (easily) deny issuing a signature
  - Crucial for legal applications
  - Judge can verify signature using a public copy of pk
- MACs cannot provide this functionality!
  - Without access to the key, no way to verify a tag
  - Even if receiver leaks key to judge, how can the judge verify that the key is correct?

Non-repudiation

- Signer cannot (easily) deny issuing a signature
  - Crucial for legal applications
  - Judge can verify signature using a public copy of pk
- MACs cannot provide this functionality!
  - Without access to the key, no way to verify a tag
  - Even if receiver leaks key to judge, how can the judge verify that the key is correct?
    - Even if key is correct, receiver could have generated the tag also!

Security

- Threat model
  - “*Adaptive chosen-message attack*”
  - Assume the attacker can induce the sender to sign *messages of the attacker’s choice*

Security

- Threat model
  - "*Adaptive chosen-message attack*"
  - Assume the attacker can induce the sender to sign *messages of the attacker's choice*

- Security goal
  - "*Existential unforgeability*"
  - Attacker should be unable to forge valid signature on any message not signed by the sender

Security

- Threat model
  - "*Adaptive chosen-message attack*"
  - Assume the attacker can induce the sender to sign *messages of the attacker's choice*

- Security goal
  - "*Existential unforgeability*"
  - Attacker should be unable to forge valid signature on any message not signed by the sender

- Attacker gets the public key

- **Definition 14.1** Fix A, Π. Define randomized experiment $Forge_{A,\Pi}(n)$:
  1. $pk, sk \leftarrow Gen(1^n)$
  2. A is given pk, and interacts with *oracle* $Sign_{sk}(n)$; let M be the set of messages sent to this oracle
  3. A outputs (m, σ)
  4. A *succeeds*, and the experiment evaluates to 1, if $Vrfy_{pk}(m, \sigma) = 1$ and $m \notin M$

  Π is *secure* if for all PPT attackers A, there is a negligible function ϵ such that

$$\Pr[Forge_{A,\Pi}(n) = 1] \leq \epsilon(n)$$

Replay attacks

- Replay attacks need to be addressed just as in the symmetric-key setting

Replay attacks

- Replay attacks need to be addressed just as in the symmetric-key setting
  - The *hash-and-sign paradigm*

Replay attacks

- Replay attacks need to be addressed just as in the symmetric-key setting
  - The *hash-and-sign paradigm*
- Given
  - A signature scheme $\Pi = (Gen, Sign, Vrfy)$ for “short” messages of length n
  - Hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$

Replay attacks

- Replay attacks need to be addressed just as in the symmetric-key setting
  - The *hash-and-sign paradigm*
- Given
  - A signature scheme $\Pi = (Gen, Sign, Vrfy)$ for "short" messages of length n
  - Hash function $H : \{0,1\}^* \to \{0,1\}^n$
- Construct a signature scheme $\Pi' = (Gen, Sign', Vrfy')$ for arbitrary-length messages:
  - $Sign'_{sk}(m) = Sign_{sk}(H(m))$
  - $Vrfy'_{pk}(m, \sigma) = Vrfy_{pk}(H(m), \sigma)$

Hash-and-sign paradigm

- **Theorem 14.2** If Π is *secure* and H is *collision-resistant*, then Π' is *secure*.

Hash-and-sign paradigm

- **Theorem 14.2** If Π is *secure* and H is *collision-resistant*, then Π' is *secure*.

  Proof. Say the sender authenticates $m_1, m_2, \ldots$
  - Let $h_i = H(m_i)$

Hash-and-sign paradigm

- **Theorem 14.2** If Π is *secure* and H is *collision-resistant*, then Π' is *secure*.

  Proof. Say the sender authenticates $m_1, m_2, \ldots$
  – Let $h_i = H(m_i)$

  Attacker outputs forgery (m, σ), $m \neq m_i$ for all i

Hash-and-sign paradigm

- **Theorem 14.2** If Π is *secure* and H is *collision-resistant*, then Π' is *secure*.

  Proof. Say the sender authenticates $m_1, m_2, \ldots$
  – Let $h_i = H(m_i)$

  Attacker outputs forgery (m, σ), $m \neq m_i$ for all i
  Two cases:
  – $H(m) = h_i$ for some i
    – Collision in H!
  – $H(m) \neq h_i$ for all i
    – Forgery in the underlying signature scheme!

"Plain" RSA signatures

$\xleftarrow{\quad N,\ e \quad}$

$\xleftarrow{\quad m,\ \sigma \quad}$

$m \stackrel{?}{=} [\sigma^e \bmod N]$

$(N, e, d) \leftarrow \mathrm{RSAGen}(1^n)$

$pk = (N, e)$

$sk = d$

$\sigma = [m^d \bmod N]$

Key generation: choose two random p, q and compute $N = p \cdot q$. Run $GenRSA(1^n)$. The secret key is (N, e). The public key is (N, d).

Signing: To sign a message m, output $\sigma = m^d \pmod{n}$.

Verification: To verify that σ is a valid signature for m, check whether $\sigma^e = m \pmod{n}$.

Security

- Intuiation
  - Signature of m is the e^{th} root of m – supposedly hard to compute

Security

- Intuiation
  - Signature of m is the e^{th} root of m – supposedly hard to compute
- Attack1: Can sign *specific* messages
  - E.g., easy to compute the e^{th} root of $m = 1$, or the cube root of $m = 8$

Security

- Intuiation
  - Signature of m is the e^{th} root of m – supposedly hard to compute
- Attack1: Can sign *specific* messages
  - E.g., easy to compute the e^{th} root of $m = 1$, or the cube root of $m = 8$
- Attack2: Can sign "*random*" messages
  - Choose arbitrary σ; set $m = \sigma^e \pmod{N}$

Security

- Intuiation
  - Signature of m is the e^{th} root of m – supposedly hard to compute
- Attack1: Can sign *specific* messages
  - E.g., easy to compute the e^{th} root of $m = 1$, or the cube root of $m = 8$
- Attack2: Can sign "*random*" messages
  - Choose arbitrary σ; set $m = \sigma^e \pmod N$
- Attack3: Can combine two signatures to obtain a third
  - Say σ_1, σ_2 are valid signatures on m_1, m_2 w.r.t. public key N, e
  - Then $\sigma' = \sigma_1 \cdot \sigma_2 \bmod N$ is a valid signature on the message $m' = m_1 \cdot m_2 \bmod N$

RSA-FDH Signature Scheme

- Main idea: apply a "*cryptographic transformation*" to messages before signing

RSA-FDH Signature Scheme

- Main idea: apply a "*cryptographic transformation*" to messages before signing

- **Construction 14.3**: Construct a signature scheme as follows:
  - *Gen*: on input 1^n, run $GenRSA(1^n)$ to compute (N, e, d). The public key is (N, e), and the private key is d. As part of key generation, a function $H : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ is specified.
  - $Sign_{sk}(m)$: on input a private key (N, d) and a message $m \in \{0,1\}^*$, compute $Sign_{sk}(m) = \sigma = H(m)^d \bmod N$
  - $Vrfy_{pk}(m, \sigma)$: On input a public key (N, e), a message m, and a signature σ, output 1 if and only if $\sigma^e = H(m) \bmod N$

Security

- Look at the three previous attacks:
  - Not easy to compute the e^{th} root of $H(1)$ ...

Security

- Look at the three previous attacks:
  - Not easy to compute the e^{th} root of $H(1)$ ...
  - Choose σ, but how do you find an m s.t. $H(m) = \sigma^e \bmod N$?
    - Computing *inverses* of H should be hard

Security

- Look at the three previous attacks:
  - Not easy to compute the e^{th} root of $H(1)$ ...
  - Choose σ, but how do you find an m s.t. $H(m) = \sigma^e \bmod N$?
    - Computing *inverses* of H should be hard
  - $H(m_1) \cdot H(m_2) = \sigma_1^e \cdot \sigma_2^e = (\sigma_1 \cdot \sigma_2)^e \neq H(m_1 \cdot m_2)$

Security

- Look at the three previous attacks:
  - Not easy to compute the e^{th} root of $H(1)$ ...
  - Choose σ, but how do you find an m s.t. $H(m) = \sigma^e \bmod N$?
    - Computing *inverses* of H should be hard
  - $H(m_1) \cdot H(m_2) = \sigma_1^e \cdot \sigma_2^e = (\sigma_1 \cdot \sigma_2)^e \neq H(m_1 \cdot m_2)$

- **Theorem 14.4** If the *RSA assumption* holds, and H is modeled as a *random oracle* (mapping onto $\mathbb{Z}_N^*$), then RSA-FDH is secure.

RSA-FDH in practice

- In practice, H is instantiated with a modified cryptographic hash function
  - Must ensure that the range of H is large enough

RSA-FDH in practice

- In practice, H is instantiated with a modified cryptographic hash function
  - Must ensure that the range of H is large enough

- The RSA PKCS #1 v2.1 standard includes a signature scheme inspired by RSA-FDH
  - Essentially a randomized variant of RSA-FDH

RSA-FDH in practice

- In practice, H is instantiated with a modified cryptographic hash function
  - Must ensure that the range of H is large enough
- The RSA PKCS $\#1$ v2.1 standard includes a signature scheme inspired by RSA-FDH
  - Essentially a randomized variant of RSA-FDH
- DSS: NIST standard for digital signatures
  - DSA, based on *discrete-logarithm problem* in subgroup of $\mathbb{Z}_p^*$
  - ECDSA, based on elliptic-curve groups

"Plain" Rabin signatures

- **Key generation**: choose two random p, q with $p, q \equiv 3 \pmod 4$, as secret keys. The public key is $n = p \cdot q$.

  Signing: To sign a message m, output $\sigma = \sqrt{m} \pmod n$ (fix some choice for one of the four possible roots).

  Verification: To verify that σ is a valid signature for m, check whether $\sigma^2 = m \pmod n$.

"Plain" Rabin signatures

- **Key generation**: choose two random p, q with $p, q \equiv 3 \pmod 4$, as secret keys. The public key is $n = p \cdot q$.

  Signing: To sign a message m, output $\sigma = \sqrt{m} \pmod n$ (fix some choice for one of the four possible roots).

  Verification: To verify that σ is a valid signature for m, check whether $\sigma^2 = m \pmod n$.

- **Note**: Assuming the *factoring problem* is hard, if m is chosen at random, then it should be hard to forge a signature for m.

“Plain” Rabin signatures

- **Key generation**: choose two random p, q with $p, q \equiv 3 \pmod 4$, as secret keys. The public key is $n = p \cdot q$.

  Signing: To sign a message m, output $\sigma = \sqrt{m} \pmod n$ (fix some choice for one of the four possible roots).

  Verification: To verify that σ is a valid signature for m, check whether $\sigma^2 = m \pmod n$.

- **Note**: Assuming the *factoring problem* is hard, if m is chosen at random, then it should be hard to forge a signature for m.

- However, this scheme is *insecure* against *chosen-message attack*.
  - Choose an $x \in \mathbb{Z}_n^*$ at random, and let $m = x^2 \pmod n$
  - Given $\sigma = \sqrt{m} \pmod n$ there is probablity $1/2$ that $\sigma \neq \pm x \pmod n$ in which case $\gcd(\sigma - x, n)$ will yield a nontrivial factor of n

Zero knowledge proofs

- **Problem**: I know that P is true, and I want to convince you of that. I try to present all the facts I know and the inferences from the facts imply P is true

Zero knowledge proofs

- **Problem**: I know that P is true, and I want to convince you of that. I try to present all the facts I know and the inferences from the facts imply P is true

- **Example**: $P:$ 26781 is not a prime since $26781 = 113 \times 237$

Zero knowledge proofs

- **Problem**: I know that P is true, and I want to convince you of that. I try to present all the facts I know and the inferences from the facts imply P is true

- **Example**: P : 26781 is not a prime since $26781 = 113 \times 237$

- Given this factorization, other than that you are convinced that P is true, you gained some knowledge (the factorization)

Zero knowledge proofs

- **Problem**: I know that P is true, and I want to convince you of that. I try to present all the facts I know and the inferences from the facts imply P is true

- **Example**: P : 26781 is not a prime since $26781 = 113 \times 237$

- Given this factorization, other than that you are convinced that P is true, you gained some knowledge (the factorization)

- In a *Zero Knowledge Proof*, Alice will prove to Bob that a statement P is true. Bob will be completely convinced that P is true, but will not learn anything as a result of this process. That is, Bob will gain zero knowledge

Zero knowledge proofs

- S. Goldwasser, S. Micali, C. Rackoff, STOC'85

The Knowledge Complexity of Interactive Proof-Systems

(Extended Abstract)

Shafi Goldwasser, MIT

Silvio Micali, MIT

Charles Rackoff, University of Toronto

Zero knowledge proofs

- S. Goldwasser, S. Micali, C. Rackoff, STOC'85

The Knowledge Complexity of Interactive Proof-Systems

(Extended Abstract)

Shafi Goldwasser
MIT

Silvio Micali
MIT

Charles Rackoff
University of Toronto

Shafi, with Micali (and later Rackoff) [6], had been thinking for a while about expanding the traditional notion of "proof" to an interactive process in which a "prover" can convince a probabilistic "verifier" of the correctness of a mathematical proposition with overwhelming probability if and only if the proposition is correct. They called this interactive process an "interactive proof" (a name suggested by Mike Sipser). They wondered if one could prove some non-trivial statement (for example, membership of a string in a hard language) without giving away any knowledge whatsoever about why it was true. They defined that the verifier receives no knowledge from the prover if the verifier could simulate on his own the probability distribution that he obtains in interacting with the prover.The idea that "no knowledge" means simulatability was a very important contribution. They also gave the first example of these "zero knowledge interactive proofs" using quadratic residuosity. This paper won the first ACM SIGACT Gödel Prize. This zero-knowledge work led to a huge research program in the community that continues to this day, including results showing that (subject to an assumption such as the existence of one-way functions) a group of distrusting parties can compute a function of all their inputs without learning any knowledge about other people's inputs beyond that which follows from the value of the function.

`https://amturing.acm.org/award_winners/goldwasser_8627889.cfm`

Applications of ZKPs

- *Protocol design*. A *protocol* is an algorithm for interactive parties to achieve a certain goal. However, in crypto, we often want to design protocols that should achieve security even when one of the parties is "cheating". Alice can prove in *zero knowledge* that she followed the instructions.

Applications of ZKPs

- *Protocol design*. A *protocol* is an algorithm for interactive parties to achieve a certain goal. However, in crypto, we often want to design protocols that should achieve security even when one of the parties is "cheating". Alice can prove in *zero knowledge* that she followed the instructions.

Proofs that Yield Nothing But their Validity
and a Methodology of Cryptographic Protocol Design

(Extended Abstract)

Oded Goldreich	Silvio Micali	Avi Wigderson
Dept. of Computer Sc.	Lab. for Computer Sc.	Inst. of Math. and CS
Technion	MIT	Hebrew University
Haifa, Israel	Cambridge, MA 02139	Jerusalem, Israel

Applications of ZKPs

- *Identification scheme*. How should Alice prove to Bob that she is who she claimed to be? For example, how to design a control access system to the CSE dept.?

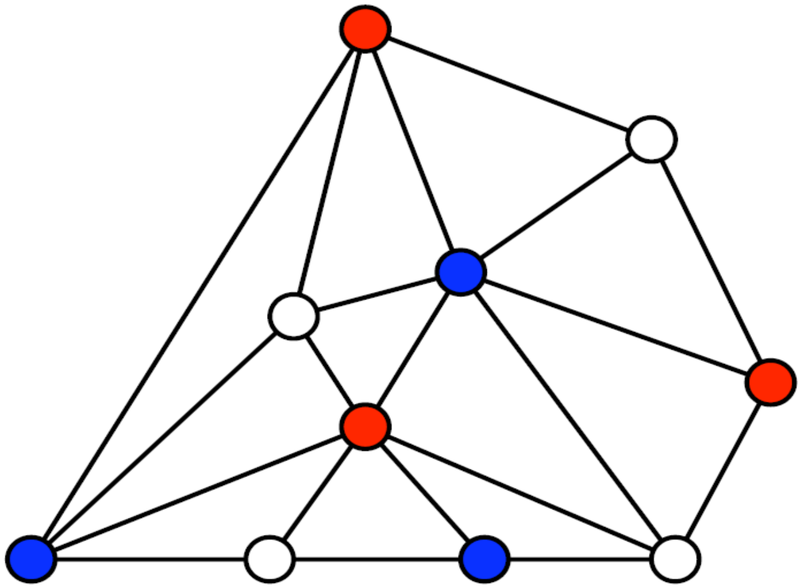
Applications of ZKPs

- *Identification scheme*. How should Alice prove to Bob that she is who she claimed to be? For example, how to design a control access system to the CSE dept.?
- A direct solution is to have a box on the door and give authorized people a secret PIN number. However, a drawback is that the box remains outside all the time and if someone could examine the box, they would perhaps be able to view its memory and extract the secrets keys of all people.

Applications of ZKPs

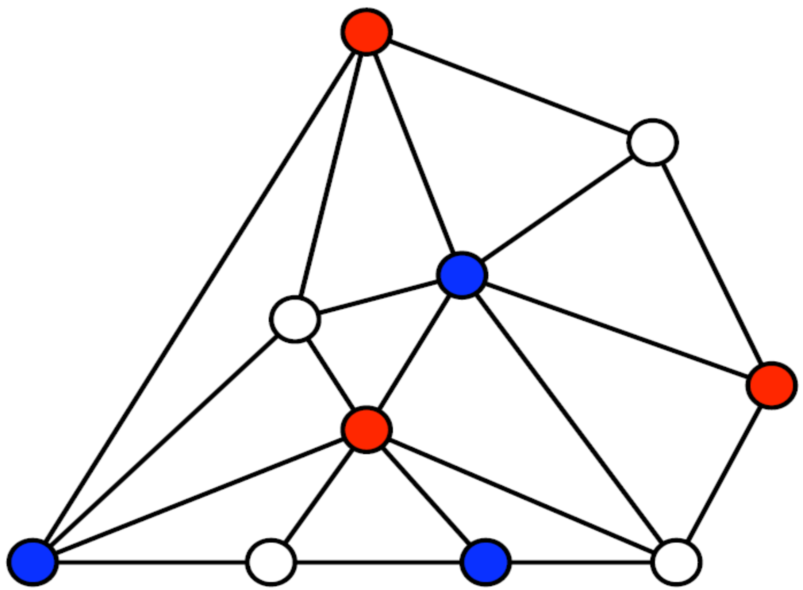
- *Identification scheme*. How should Alice prove to Bob that she is who she claimed to be? For example, how to design a control access system to the CSE dept.?
- A direct solution is to have a box on the door and give authorized people a secret PIN number. However, a drawback is that the box remains outside all the time and if someone could examine the box, they would perhaps be able to view its memory and extract the secrets keys of all people.
- **Ideas using ZKPs**:
  - Let the box contain an *instance* of a hard problem.
  - Give the authorized people the *solution* to the instance.
  - The authorized people will *prove* to the box that they know the solution in zero knowledge.

An example



- Alice knows how to 3-color a graph: no two adjacent vertices have the same color; this is an NPC problem.

An example



- Alice knows how to 3-color a graph: no two adjacent vertices have the same color; this is an NPC problem.
  - can impress your friends
  - useful for identification

An example

- How can Alice convince Bob that she can 3-color the graph without
  - letting him steal her work?
  - letting him impersonate her?

An example

- How can Alice convince Bob that she can 3-color the graph without
  - letting him steal her work?
  - letting him impersonate her?

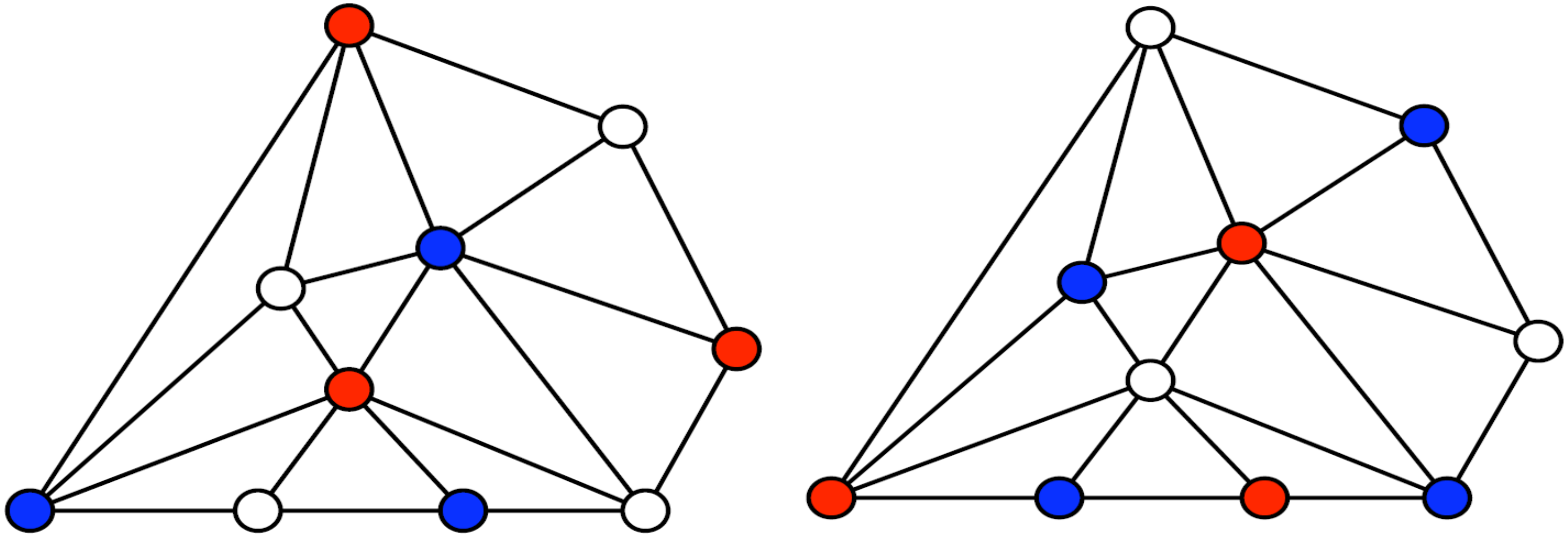
  - Bob is convinced that Alice can do this.
  - Bob has no idea how to do it himself.

An example

- How can Alice convince Bob that she can 3-color the graph without
  - letting him steal her work?
  - letting him impersonate her?

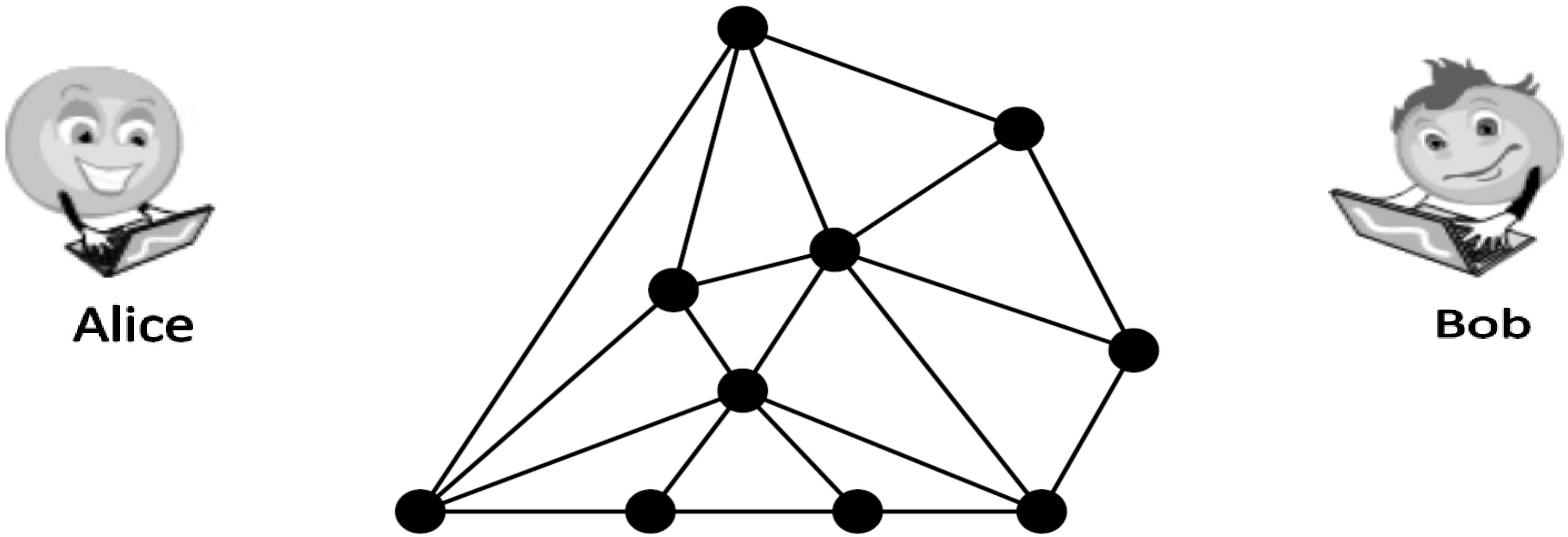
  - Bob is convinced that Alice can do this.
  - Bob has no idea how to do it himself.

Alice may permute the vertex colors.



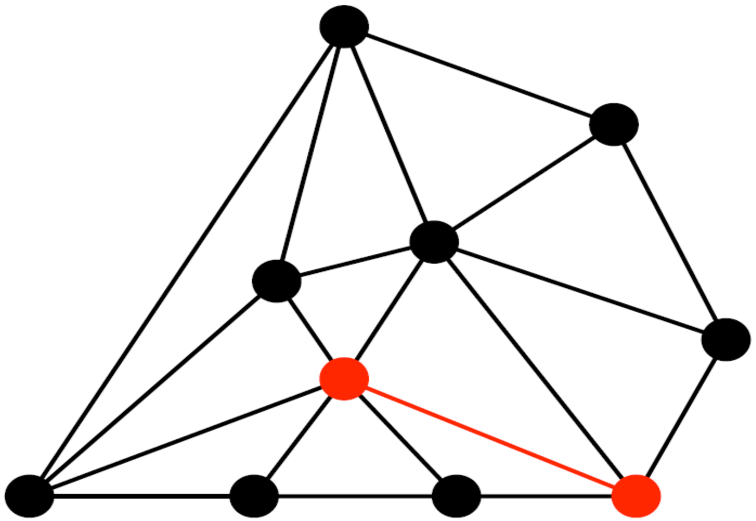
An example

- Alice then encrypts all vertex colors (one key per vertex), and sends the graph to Bob.



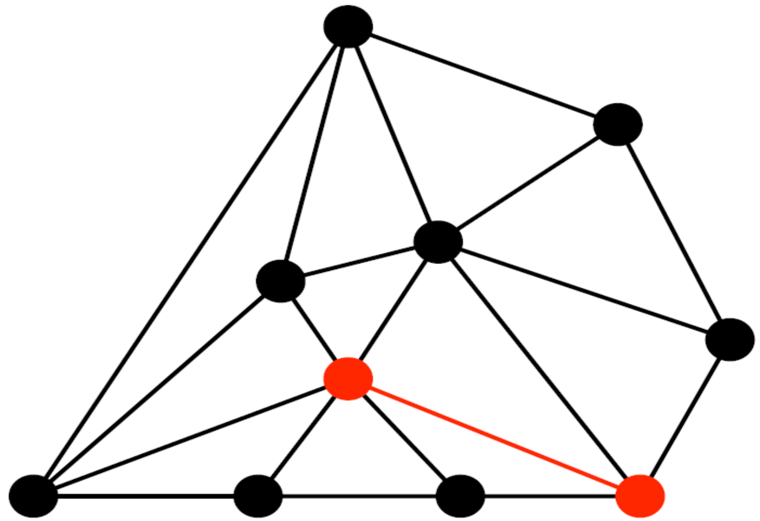
An example

- Bob picks an edge at random.

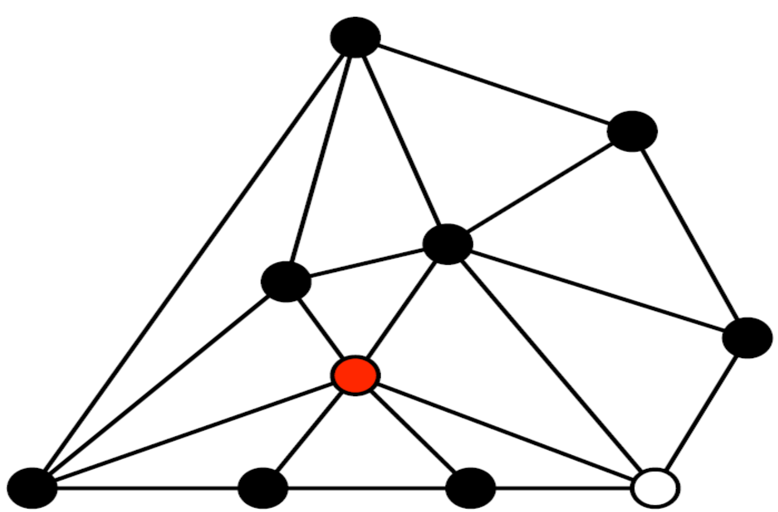


An example

- Bob picks an edge at random.



Alice reveals colors of those two keys.



An example

- Repeat as much as needed:
  - Alice permutes graph coloring
  - Alice encrypts all vertices with distinct keys
  - Alice sends permuted encrypted colors to Bob
  - Bob picks an edge
  - Alice sends keys for two vertices
  - Bob checks whether these two colors are distinct

An example

- Repeat as much as needed:
  - Alice permutes graph coloring
  - Alice encrypts all vertices with distinct keys
  - Alice sends permuted encrypted colors to Bob
  - Bob picks an edge
  - Alice sends keys for two vertices
  - Bob checks whether these two colors are distinct

If Alice is lying, with probability $\frac{1}{|E|}$ she will be caught.
If she is telling the truth, she will never be caught.

An example

- Repeat as much as needed:
  - Alice permutes graph coloring
  - Alice encrypts all vertices with distinct keys
  - Alice sends permuted encrypted colors to Bob
  - Bob picks an edge
  - Alice sends keys for two vertices
  - Bob checks whether these two colors are distinct

If Alice is lying, with probability $\frac{1}{|E|}$ she will be caught.
If she is telling the truth, she will never be caught.

After k repetitions, the probability she fools Bob is $(1 - \frac{1}{|E|})^k$.

An example

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

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.

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.

Claim. Every NP-statement can be proven in zero-knowledge.

Zero knowledge proofs

- A standard mathematical proof is like:

  Given *axioms* and *inference rules*, we give the proof for P that derives P from the axioms using the inference rules.

- A standard mathematical proof is like:

  Given *axioms* and *inference rules*, we give the proof for P that derives P from the axioms using the inference rules.

- A proof system is *sound* if you can never derive **false** statements using it.

  A proof system is *complete* if you can prove all **true** statements using it.

Zero knowledge proofs

- A standard mathematical proof is like:

  Given *axioms* and *inference rules*, we give the proof for P that derives P from the axioms using the inference rules.

- A proof system is *sound* if you can never derive **false** statements using it.

  A proof system is *complete* if you can prove all **true** statements using it.

- *Interactive probabilistic proofs*: the verifier does not convince with absolute certainty that P is true, but with high certainty.

Interactive Probabilistic Proofs

- *Interactive probabilistic proofs*: the verifier does not convince with absolute certainty that P is true, but with high certainty.

  No matter what the prover does, and how she tries to to cheat, if the statement P is false, she will fail with this probability.

Interactive Probabilistic Proofs

- *Interactive probabilistic proofs*: the verifier does not convince with absolute certainty that P is true, but with high certainty.

  No matter what the prover does, and how she tries to to cheat, if the statement P is false, she will fail with this probability.

- **Example.** Alice can distinguish between Coke and Pepsi:
  Alice turns her back, Bob flips a coin and puts either Coke and Pepsi into a paper cup according the result, Alice tastes and announces whether she thinks it is Coke or Pepsi.

Interactive Probabilistic Proofs

- *Interactive probabilistic proofs*: the verifier does not convince with absolute certainty that P is true, but with high certainty.

  No matter what the prover does, and how she tries to to cheat, if the statement P is false, she will fail with this probability.

- **Example.** Alice can distinguish between Coke and Pepsi:
  Alice turns her back, Bob flips a coin and puts either Coke and Pepsi into a paper cup according the result, Alice tastes and announces whether she thinks it is Coke or Pepsi.

- If they repeat this k times, and Alice always answers correctly, then Bob can conclude with $1 - 2^{-k}$ probability that she really can tell the difference.

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**:

  ◇ 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.

  ◇ 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
  Prover'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
  Prover'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
  Prover's private input: w such that $x = w^2 \pmod{n}$

$P \to V$: Alice chooses random $u \leftarrow_R \mathbb{Z}_n^*$ and sends $y = u^2$ to Bob
$P \leftarrow V$: Bob chooses $b \leftarrow_R \{0, 1\}$
$P \to V$: 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
  Prover'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 s such that $x = s^2 \pmod{n}$, 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 s such that $x = s^2 \pmod{n}$, 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 s such that $x = s^2 \pmod{n}$, 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 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 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.

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

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

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.

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 $x \notin QR_n$.

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 $\geq 1/2$.

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:
  - $\mathit{view}_{V^*}\langle P_{U_m,x,w}, V^*\rangle$, 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 $h^{-b}g^c$.

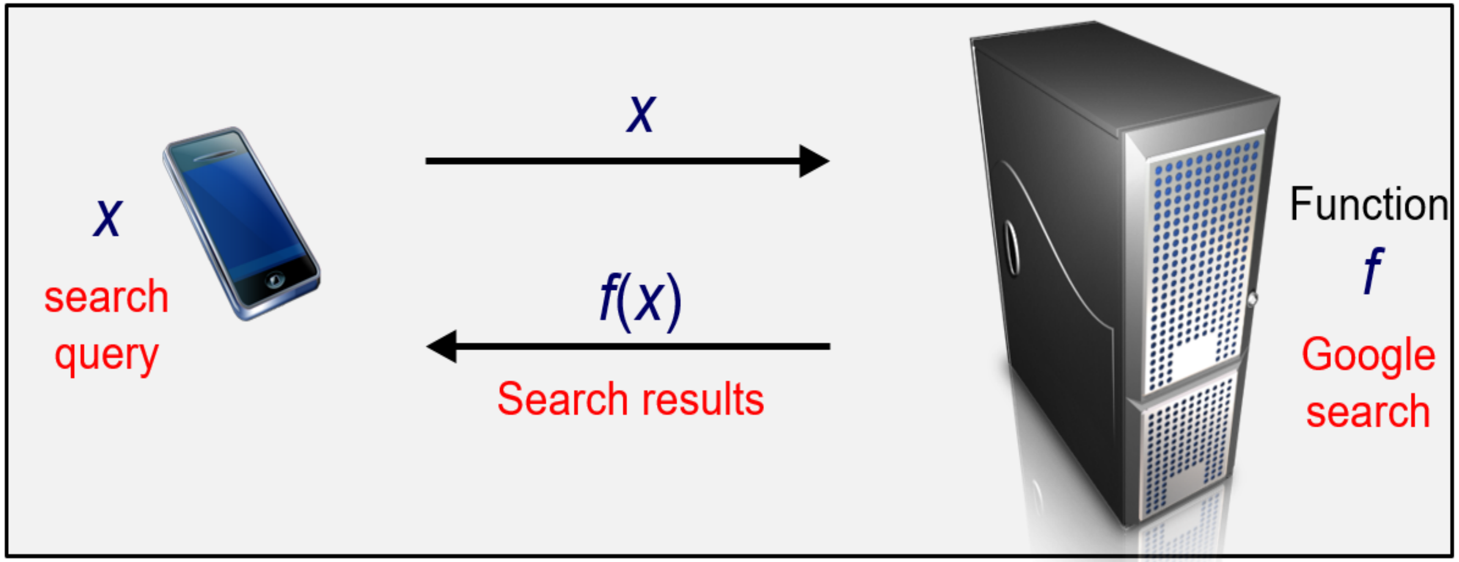
- **Definition 15.6** (*Group homomorphism*)
  Two groups G and G' are *homomorphic* if there exists a function (*homomorphism*) $f : G \to G'$ such that for all $x, y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$.

- **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*?

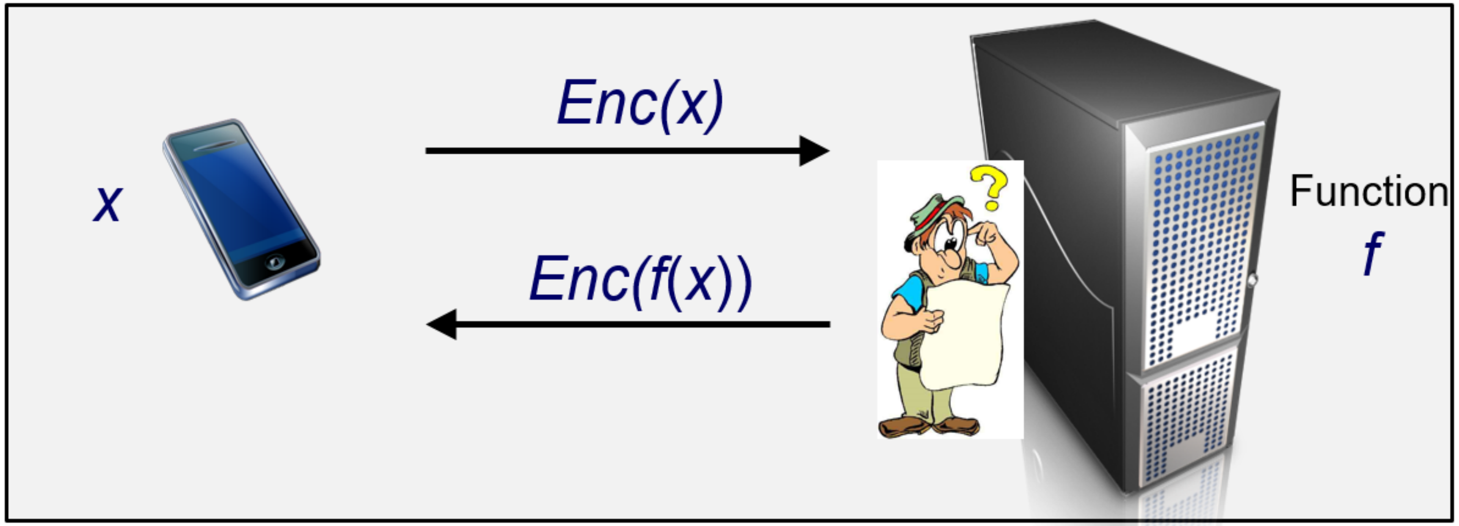
- **Definition 15.6** (*Group homomorphism*)
  Two groups G and G' are *homomorphic* if there exists a function (*homomorphism*) $f : G \to G'$ such that for all $x, y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$.

  Why do we need *homomorphic encryption*?



- **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**!

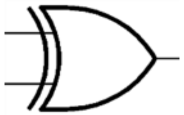
  What people really wanted was the ability to do arbitrary computing on encrypted data, and this requires the abibility to compute both sums and products.

Computing on encrypted data

- Why SUMs and PRODUCTs?

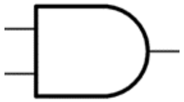
- Why SUMs and PRODUCTs?

SUM



XOR

PRODUCT

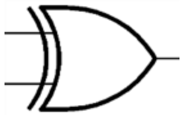


AND

Computing on encrypted data

- Why SUMs and PRODUCTs?

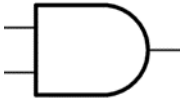
SUM



XOR

$x + y \bmod 2$

PRODUCT



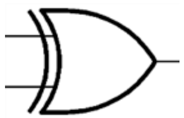
AND

$x \cdot y \bmod 2$

Computing on encrypted data

- Why SUMs and PRODUCTs?

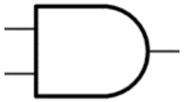
SUM



XOR

$x + y \bmod 2$

PRODUCT



AND

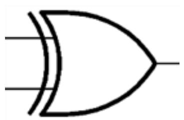
$x \cdot y \bmod 2$

{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?

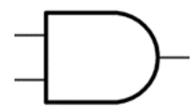
SUM



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

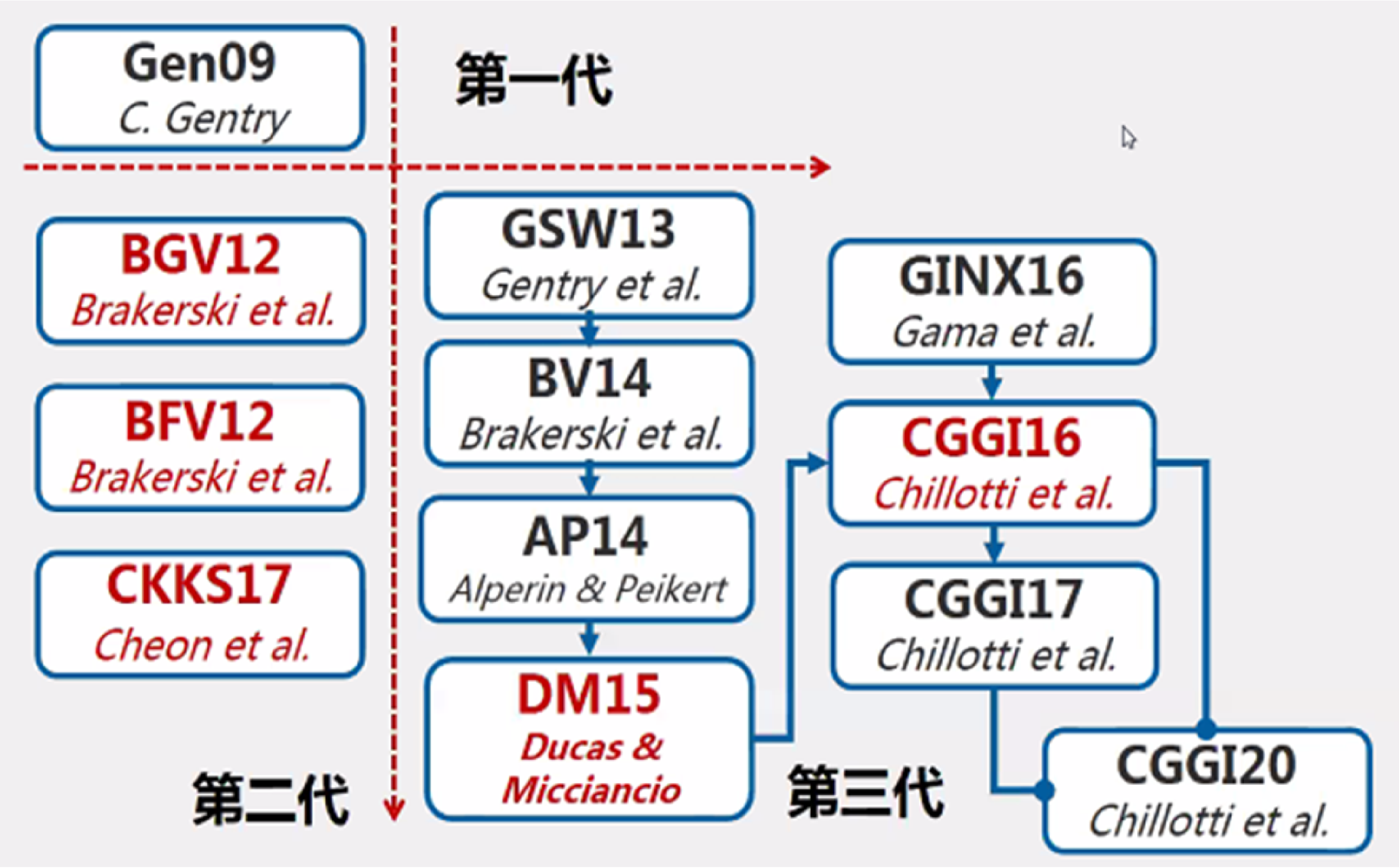
Fully Homomorphic Encryption over the Integers

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

Post-Quantum Cryptography (PQC)

Post-Quantum Cryptography (PQC)

PQC Standardization Process: Third Round Candidate Announcement

July 22, 2020

PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates

July 05, 2022

Post-Quantum Safe Algorithm Candidate Cracked in an Hour on a PC

BY MATT SWAYNE • AUGUST 5, 2022 • RESEARCH

Good Luck!