OSY.SSI[2018][8]

In the news...

- TODO

Please prepare for the lab

Bring your laptop.

We will try something new that may or may not work.

Also install WireShark + PuTTY (or `ssh`) on your *host* device.

In the previous episode...

$\emptyset$ · DiD · OTP/IND-1T

$\bullet_0$ $\bullet_1$ $\bullet_2$ $\bullet_3$ $\bullet_4$ $\bullet_5$ $\bullet_6$ $\bullet_7$ $\bullet_8$ $\bullet_9$ $\bullet_{10}$ $\bullet_{11}$ $\bullet_{12}$ $\bullet_{13}$ $\bullet_{14}$ $\bullet_{15}$ EXAM

The issue with OTP

The issue with OTP

- On the one hand we have provably perfect Shannon security.

The issue with OTP

- On the one hand we have provably perfect Shannon security.
- On the other hand it is very impractical

The issue with OTP

- On the one hand we have provably perfect Shannon security.
- On the other hand it is very impractical
- And in any case it only works for confidentiality

Today's lecture

Today's lecture

- Trade security for practicability: replace "impossible" by "hard"

Today's lecture

- Trade security for practicability: replace "impossible" by "hard"
- Address data integrity (By defining a game)

Today's lecture

- Trade security for practicability: replace "impossible" by "hard"
- Address data integrity (By defining a game)
- See what we need to establish a secure channel

Table of Contents

Statistics vs. complexity

Maybe instead of an “impossible to break” system, we just need “hard to break” systems.

What do we mean, “hard”?

Complexity

Algorithmic time complexity

An algorithm is said to have time complexity $O(f(n))$ if, given an input of size n, this algorithm terminates in time $O(f(n))$.

Complexity

Algorithmic time complexity

An algorithm is said to have time complexity $O(f(n))$ if, given an input of size n, this algorithm terminates in time $O(f(n))$.

Extended Church–Turing thesis: It does not make a huge difference how the algorithm is implemented and on what device it runs.

Complexity

Algorithmic time complexity

An algorithm is said to have time complexity $O(f(n))$ if, given an input of size n, this algorithm terminates in time $O(f(n))$.

Extended Church–Turing thesis: It does not make a huge difference how the algorithm is implemented and on what device it runs.

Examples:

- Finding the max element in a list of size n can be done in $O(n)$.
- Sorting a list of size n can be done in $O(n \log n)$.

P vs. NP

$\mathbf{P}$ and $\mathbf{NP}$

- $\mathbf{PTIME}$, or simply $\mathbf{P}$: Problems for which we can find an exact solution in time $O(\text{poly})$.

P vs. NP

P and NP

- **PTIME**, or simply **P**: Problems for which we can find an exact solution in time $O(\text{poly})$.
- **NPTIME**, or simply **NP** : Problems for which we can check a solution in time $O(\text{poly})$

P vs. NP

$\mathbf{P}$ and $\mathbf{NP}$

- $\mathbf{PTIME}$, or simply **P**: Problems for which we can find an exact solution in time $O(\text{poly})$.
- $\mathbf{NPTIME}$, or simply **NP** : Problems for which we can check a solution in time $O(\text{poly})$
- In particular, $\mathbf{P} \subset \mathbf{NP}$.

Three remarks:

- Problems in P or NP are always computable
- P and NP are well-defined thanks to the ECTT
- **P, NP $\subseteq$ PSPACE**

P and NP problems

Problems in $\mathbf{P}$:

- linear programming, greatest common divisor
- Type inference
- Determining if a number is prime

Problems in $\mathbf{NP}$:

- Hamiltonian path
- Traveling salesman problem
- Knapsack / Subset-sum

General intuition: $\mathbf{P}$ is "easy", $\mathbf{NP}$ is "interesting"

There are many, easier, harder, or unrelated classes: **L**, **PP**, **NP-hard**, **BQP**, **AM**, **#P**, **EXPTIME**, etc.

A strange asymmetry

For reasons that are not yet fully clear, there seems to be a difference between

- Checking that a given solution to a problem works, and
- Finding a solution to that problem from scratch

Example:

A strange asymmetry

For reasons that are not yet fully clear, there seems to be a difference between

- Checking that a given solution to a problem works, and
- Finding a solution to that problem from scratch

Example: curing cancer.

A strange asymmetry

For reasons that are not yet fully clear, there seems to be a difference between

- Checking that a given solution to a problem works, and
- Finding a solution to that problem from scratch

Example: curing cancer.

Exercice: Prove that $\mathbf{P} = \mathbf{NP}$ or find a counter-example.

A strange asymmetry

Based on the observation that finding and checking a solution seem different, consider the following question:

Is there a function $f : X \to Y$ such that:

A strange asymmetry

Based on the observation that finding and checking a solution seem different, consider the following question:

Is there a function $f : X \to Y$ such that:

1. For any $x \in X$, computing $f(x)$ is easy, but

A strange asymmetry

Based on the observation that finding and checking a solution seem different, consider the following question:

Is there a function $f : X \to Y$ such that:

1. For any $x \in X$, computing $f(x)$ is easy, but
2. For any $y \in Y$, finding an x such that $f(x) = y$ is hard?

A strange asymmetry

Based on the observation that finding and checking a solution seem different, consider the following question:

Is there a function $f : X \to Y$ such that:

1. For any $x \in X$, computing $f(x)$ is easy, but
2. For any $y \in Y$, finding an x such that $f(x) = y$ is hard?

We don't know! This would entail $\mathbf{P} \neq \mathbf{NP}$ (which we don't know)

A strange asymmetry

Based on the observation that finding and checking a solution seem different, consider the following question:

Is there a function $f : X \to Y$ such that:

1. For any $x \in X$, computing $f(x)$ is easy, but
2. For any $y \in Y$, finding an x such that $f(x) = y$ is hard?

We don't know! This would entail $\mathbf{P} \neq \mathbf{NP}$ (which we don't know)

But we can try.

Trying to get a one-way function

Given two integers p and q, it is easy to compute the product $p \times q$.

Example: $12345 \times 100000 = 1234500000$.

Trying to get a one-way function

Given two integers p and q, it is easy to compute the product $p \times q$.

Example: $12345 \times 100000 = 1234500000$. $\approx O(|p| \times |q|)$

Trying to get a one-way function

Given two integers p and q, it is easy to compute the product $p \times q$.

Example: $12345 \times 100000 = 1234500000$. $\approx O(|p| \times |q|)$

However, given $n = p \times q$, how to recover p and q?

Example: $1234500001 =$

Trying to get a one-way function

Given two integers p and q, it is easy to compute the product $p \times q$.

Example: $12345 \times 100000 = 1234500000$. $\approx O(|p| \times |q|)$

However, given $n = p \times q$, how to recover p and q?

Example: $1234500001 = 7 \times 176357143$.

Trying to get a one-way function

Given two integers p and q, it is easy to compute the product $p \times q$.

Example: $12345 \times 100000 = 1234500000$. $\approx O(|p| \times |q|)$

However, given $n = p \times q$, how to recover p and q?

Example: $1234500001 = 7 \times 176357143$.

The best known classical generic algorithm for factorisation is the generalised number field sieve (GNFS), with a heuristic complexity of

Trying to get a one-way function

Given two integers p and q, it is easy to compute the product $p \times q$.

Example: $12345 \times 100000 = 1234500000$. $\approx O(|p| \times |q|)$

However, given $n = p \times q$, how to recover p and q?

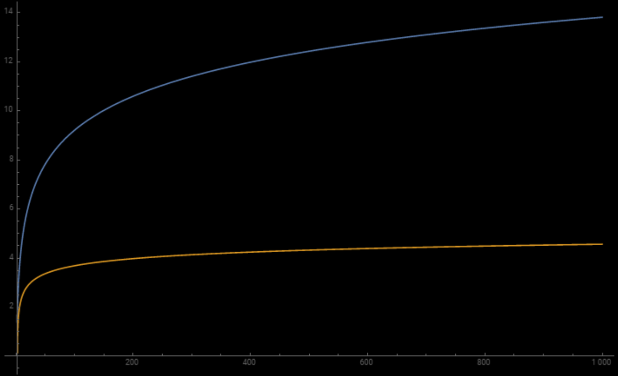
Example: $1234500001 = 7 \times 176357143$.

The best known classical generic algorithm for factorisation is the generalised number field sieve (GNFS), with a heuristic complexity of

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}\right)$$

Trying to get a one-way function

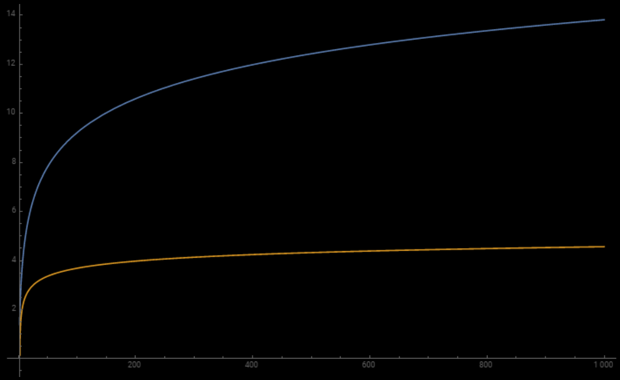
Difficulty (log scale) as a function of n:



Orange: cost of multiplication, Blue: cost of factorisation.

Trying to get a one-way function

Difficulty (log scale) as a function of n:



Orange: cost of multiplication, Blue: cost of factorisation.

So for instance, if we have $p, q > 2^{1600}$, the cost of factorisation is around... 2^{128}.

Another candidate?

Another candidate?

Consider a cyclic group $G = \langle g \rangle = \{1, g, g^2, g^3, g^4, \ldots, g^{p-1}\}$

Example: $G = \mathbb{F}_{17}^{\times} = \{3^k \bmod 17\} = \{1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6\}$

(Other examples: $\mathbb{F}_{p^k}^{\times}$, $E(\mathbb{F}_{p^k})$.)

Computing g^k for any k is easy

Example: $g^{12345} = g^{12345 \bmod 16} = g^9 = 14$ $\qquad O(\log k) = O(\log p)$

Given $u \in G$, finding a k such that $u = g^k$ ...

Another candidate?

Consider a cyclic group $G = \langle g \rangle = \{1, g, g^2, g^3, g^4, \dots, g^{p-1}\}$

Example: $G = \mathbb{F}_{17}^{\times} = \{3^k \bmod 17\} = \{1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6\}$

(Other examples: $\mathbb{F}_{p^k}^{\times}$, $E(\mathbb{F}_{p^k})$.)

Computing g^k for any k is easy

Example: $g^{12345} = g^{12345 \bmod 16} = g^9 = 14$ $\qquad O(\log k) = O(\log p)$

Given $u \in G$, finding a k such that $u = g^k$ …

- Assume p is prime

Another candidate?

Consider a cyclic group $G = \langle g \rangle = \{1, g, g^2, g^3, g^4, \ldots, g^{p-1}\}$

Example: $G = \mathbb{F}_{17}^\times = \{3^k \bmod 17\} = \{1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6\}$

(Other examples: $\mathbb{F}_{p^k}^\times$, $E(\mathbb{F}_{p^k})$.)

Computing g^k for any k is easy

Example: $g^{12345} = g^{12345 \bmod 16} = g^9 = 14$ $\qquad O(\log k) = O(\log p)$

Given $u \in G$, finding a k such that $u = g^k$ ...

- Assume p is prime
- If $G = \mathbb{F}_p^\times$ the best known classical algorithm is the GNFS $\qquad L_p(1/3, (64/9)^{2/3})$

Another candidate?

Consider a cyclic group $G = \langle g \rangle = \{1, g, g^2, g^3, g^4, \dots, g^{p-1}\}$

Example: $G = \mathbb{F}_{17}^{\times} = \{3^k \bmod 17\} = \{1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6\}$

(Other examples: $\mathbb{F}_{p^k}^{\times}$, $E(\mathbb{F}_{p^k})$.)

Computing g^k for any k is easy

Example: $g^{12345} = g^{12345 \bmod 16} = g^9 = 14$ $\qquad O(\log k) = O(\log p)$

Given $u \in G$, finding a k such that $u = g^k$ ...

- Assume p is prime
- If $G = \mathbb{F}_p^{\times}$ the best known classical algorithm is the GNFS $\qquad L_p(1/3, (64/9)^{2/3})$
- If $G = E(\mathbb{F}_p)$ the best known classical algorithm is Pollard's ρ $\qquad O(\sqrt{p})$

Candidate hard problems in crypto

We already have

- Integer factorisation: Given $n \in \mathbb{N}$, find d such that $d|n$.
- Discrete logarithm: Given $h \in \langle g \rangle$, find x such that $h = g^x$.

Candidate hard problems in crypto

We already have

- Integer factorisation: Given $n \in \mathbb{N}$, find d such that $d|n$.
- Discrete logarithm: Given $h \in \langle g \rangle$, find x such that $h = g^x$.

We can choose n and G to make the problem *arbitrarily hard*.

Candidate hard problems in crypto

We already have

- Integer factorisation: Given $n \in \mathbb{N}$, find d such that $d|n$.
- Discrete logarithm: Given $h \in \langle g \rangle$, find x such that $h = g^x$.

We can choose n and G to make the problem *arbitrarily hard*.

This also increases the cost of multiplication (resp. exponentiation)

Candidate hard problems in crypto

We already have

- Integer factorisation: Given $n \in \mathbb{N}$, find d such that $d|n$.
- Discrete logarithm: Given $h \in \langle g \rangle$, find x such that $h = g^x$.

We can choose n and G to make the problem *arbitrarily hard*.

This also increases the cost of multiplication (resp. exponentiation)

So we want to make it as hard to invert f while keeping it easy to compute f.

Candidate hard problems in crypto

We already have

- Integer factorisation: Given $n \in \mathbb{N}$, find d such that $d|n$.
- Discrete logarithm: Given $h \in \langle g \rangle$, find x such that $h = g^x$.

We can choose n and G to make the problem *arbitrarily hard*.

This also increases the cost of multiplication (resp. exponentiation)

So we want to make it as hard to invert f while keeping it easy to compute f.

Example: $p \approx 2^{256}$ gives a user cost of $\approx 2^8$ and an adversarial cost of $\approx 2^{128}$.

There are many other candidates out there, we focus on these two which are the most widely in use today.

Okay so what do we do now with one-way functions?

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.
- Alice computes $K = B^a$, while Bob computes $K = A^b$.

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.
- Alice computes $K = B^a$, while Bob computes $K = A^b$.

Remarks:

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.
- Alice computes $K = B^a$, while Bob computes $K = A^b$.

Remarks:

- $K = g^{ab} = g^{ba}$, in other terms, Alice and Bob have negociated K

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.
- Alice computes $K = B^a$, while Bob computes $K = A^b$.

Remarks:

- $K = g^{ab} = g^{ba}$, in other terms, Alice and Bob have negociated K
- A and B can be sent over an <u>insecure</u> communication channel, why?

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.
- Alice computes $K = B^a$, while Bob computes $K = A^b$.

Remarks:

- $K = g^{ab} = g^{ba}$, in other terms, Alice and Bob have negociated K
- A and B can be sent over an <u>insecure</u> communication channel, why?
- Computational Diffie–Hellman (CDH) problem: Given (g, g^a, g^b), compute g^{ab}.

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.
- Alice computes $K = B^a$, while Bob computes $K = A^b$.

Remarks:

- $K = g^{ab} = g^{ba}$, in other terms, Alice and Bob have negociated K
- A and B can be sent over an <u>insecure</u> communication channel, why?
- Computational Diffie–Hellman (CDH) problem: Given (g, g^a, g^b), compute g^{ab}.
- Best known algorithm requires solving the discrete logarithm problem in G.

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.
- Alice computes $K = B^a$, while Bob computes $K = A^b$.

Remarks:

- $K = g^{ab} = g^{ba}$, in other terms, Alice and Bob have negociated K
- A and B can be sent over an insecure communication channel, why?
- Computational Diffie–Hellman (CDH) problem: Given (g, g^a, g^b), compute g^{ab}.
- Best known algorithm requires solving the discrete logarithm problem in G.

Using this (DHE), Alice and Bob have shared a secret!

Okay so what do we do now with one-way functions?

In the 1970's (Cliff Cocks and) Diffie and Hellman proposed the following idea:

- Choose a group $G = \langle g \rangle$.
- Alice picks a number a, and sends $A = g^a$ to Bob.
- Bob picks a number b, and sends $B = g^b$ to Alice.
- Alice computes $K = B^a$, while Bob computes $K = A^b$.

Remarks:

- $K = g^{ab} = g^{ba}$, in other terms, Alice and Bob have negociated K
- A and B can be sent over an insecure communication channel, why?
- Computational Diffie–Hellman (CDH) problem: Given (g, g^a, g^b), compute g^{ab}.
- Best known algorithm requires solving the discrete logarithm problem in G.

Using this (DHE), Alice and Bob have shared a secret! Solving the key exchange problem!

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

- We have only computational confidentiality, no longer *perfect* (Shannon) security

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

- We have only computational confidentiality, no longer *perfect* (Shannon) security
- We need to agree on G (that's ok, Logjam though), better use elliptic curves (ECDHE)

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

- We have only computational confidentiality, no longer *perfect* (Shannon) security
- We need to agree on G (that's ok, Logjam though), better use elliptic curves (ECDHE)
- We still need random numbers a and b

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

- We have only computational confidentiality, no longer *perfect* (Shannon) security
- We need to agree on G (that's ok, Logjam though), better use elliptic curves (ECDHE)
- We still need random numbers a and b but *pseudo-random* will suffice

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

- We have only computational confidentiality, no longer *perfect* (Shannon) security
- We need to agree on G (that's ok, Logjam though), better use elliptic curves (ECDHE)
- We still need random numbers a and b but *pseudo-random* will suffice
- Should we use K directly as a key?

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

- We have only computational confidentiality, no longer *perfect* (Shannon) security
- We need to agree on G (that's ok, Logjam though), better use elliptic curves (ECDHE)
- We still need random numbers a and b but *pseudo-random* will suffice
- Should we use K directly as a key? The issue is, K may not be uniformly random.

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

- We have only computational confidentiality, no longer *perfect* (Shannon) security
- We need to agree on G (that's ok, Logjam though), better use elliptic curves (ECDHE)
- We still need random numbers a and b but *pseudo-random* will suffice
- Should we use K directly as a key? The issue is, K may not be uniformly random.
- What is an obvious attack on the protocol?

Diffie–Hellman Key Exchange Protocol

The above protocol is widely deployed and an integral part of TLS. A few remarks are in order:

- We have only computational confidentiality, no longer *perfect* (Shannon) security
- We need to agree on G (that's ok, Logjam though), better use elliptic curves (ECDHE)
- We still need random numbers a and b but *pseudo-random* will suffice
- Should we use K directly as a key? The issue is, K may not be uniformly random.
- What is an obvious attack on the protocol? We need integrity.

Cryptographic integrity

We want to mathematically capture integrity.

Cryptographic integrity

We want to mathematically capture integrity.

> *Given a message* m*, we want an algorithm that produces* $\sigma(m)$ *such that* $\sigma(m') \neq \sigma(m)$

Is that satisfactory?

Cryptographic integrity

We want to mathematically capture integrity.

> *Given a message m, we want an algorithm that produces $\sigma(m)$ such that $\sigma(m') \neq \sigma(m)$*

Is that satisfactory?

As stated, any adversary can alter the message and re-compute σ... there is no adversarial resistance. We need *signer commitment*.

Cryptographic integrity

We want to mathematically capture integrity.

> *Given a message* m*, we want an algorithm that produces* $\sigma(m)$ *such that* $\sigma(m') \neq \sigma(m)$

Is that satisfactory?

As stated, any adversary can alter the message and re-compute σ... there is no adversarial resistance. We need *signer commitment*.

In other terms, only the signatory should be able to compute σ. Can we make this desire precise?

Cryptographic integrity

Consider the following game called EF-CMA:

Adversary		Alice
Picks $m_1, \dots, m_n$	$\xrightarrow{m_1,\dots,m_n}$	
	$\xleftarrow{\sigma_1,\dots,\sigma_n}$	Signs everything
Outputs $m^\star, \sigma^\star$		

Cryptographic integrity

Consider the following game called EF-CMA:

Adversary		Alice
Picks $m_1, \ldots, m_n$	$\xrightarrow{m_1,\ldots,m_n}$	
	$\xleftarrow{\sigma_1,\ldots,\sigma_n}$	Signs everything
Outputs $m^\star, \sigma^\star$		

We say that the adversary wins this game if $m^\star$, $\sigma^\star$ is valid (and not seen before).

We say that the adversary produced an *existential forgery*.

Cryptographic integrity

Consider the following game called EF-CMA:

Adversary		Alice
Picks $m_1, \dots, m_n$	$\xrightarrow{m_1,\dots,m_n}$	
	$\xleftarrow{\sigma_1,\dots,\sigma_n}$	Signs everything
Outputs $m^\star, \sigma^\star$		

We say that the adversary wins this game if $m^\star$, $\sigma^\star$ is valid (and not seen before).

We say that the adversary produced an *existential forgery*.

Goal: design an algorithm σ such that the probability of the adversary winning is negligible!

Cryptographic integrity: Digital signatures

For a long time, we didn't know how to prove authenticity using just one-way functions.

Cryptographic integrity: Digital signatures

For a long time, we didn't know how to prove authenticity using just one-way functions.

The first digital signature scheme was proposed by Rivest, Shamir and Adleman (RSA):

Cryptographic integrity: Digital signatures

For a long time, we didn't know how to prove authenticity using just one-way functions.

The first digital signature scheme was proposed by Rivest, Shamir and Adleman (RSA):

- Choose a secret p, q and publish $n = p \times q$.
- Choose and publish e coprime with $\varphi(n)$.

Cryptographic integrity: Digital signatures

For a long time, we didn't know how to prove authenticity using just one-way functions.

The first digital signature scheme was proposed by Rivest, Shamir and Adleman (RSA):

- Choose a secret p, q and publish $n = p \times q$.
- Choose and publish e coprime with $\varphi(n)$.
- Sign by computing $\sigma = m^{1/e} \bmod n$
- Verify the signature by checking whether $\sigma^e = m \bmod n$.

Cryptographic integrity: Digital signatures

For a long time, we didn't know how to prove authenticity using just one-way functions.

The first digital signature scheme was proposed by Rivest, Shamir and Adleman (RSA):

- Choose a secret p, q and publish $n = p \times q$.
- Choose and publish e coprime with $\varphi(n)$.
- Sign by computing $\sigma = m^{1/e} \bmod n$
- Verify the signature by checking whether $\sigma^e = m \bmod n$.

These are called RSA signatures.

Cryptographic integrity: Digital signatures

For a long time, we didn't know how to prove authenticity using just one-way functions.

The first digital signature scheme was proposed by Rivest, Shamir and Adleman (RSA):

- Choose a secret p, q and publish $n = p \times q$.
- Choose and publish e coprime with $\varphi(n)$.
- Sign by computing $\sigma = m^{1/e} \bmod n$
- Verify the signature by checking whether $\sigma^e = m \bmod n$.

These are called RSA signatures.

Hypothesis: Computing e-th roots modulo n is hard (RSA hypothesis).

Cryptographic integrity: Digital signatures

For a long time, we didn't know how to prove authenticity using just one-way functions.

The first digital signature scheme was proposed by Rivest, Shamir and Adleman (RSA):

- Choose a secret p, q and publish $n = p \times q$.
- Choose and publish e coprime with $\varphi(n)$.
- Sign by computing $\sigma = m^{1/e} \bmod n$
- Verify the signature by checking whether $\sigma^e = m \bmod n$.

These are called RSA signatures.

Hypothesis: Computing e-th roots modulo n is hard (RSA hypothesis).

Best known generic algorithm requires factorisation of n.

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity
- The message m must be an integer between 2 and $n-1$...

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity
- The message m must be an integer between 2 and $n-1$...
- RSA as described above is not EF-CMA-secure:

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity
- The message m must be an integer between 2 and $n-1$...
- RSA as described above is not EF-CMA-secure:
  - Take any random value $\sigma^\star$ and compute $m^\star = (\sigma^\star)^e \bmod n$.

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity
- The message m must be an integer between 2 and $n-1$...
- RSA as described above is not EF-CMA-secure:
  - Take any random value $\sigma^\star$ and compute $m^\star = (\sigma^\star)^e \bmod n$.
  - Furthermore, given $m_1, m_2, \sigma_1, \sigma_2$, we can let $\sigma^\star = \sigma_1\sigma_2$ and $m^\star = m_1 m_2$.

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity
- The message m must be an integer between 2 and $n-1$...
- RSA as described above is not EF-CMA-secure:
  - Take any random value $\sigma^\star$ and compute $m^\star = (\sigma^\star)^e \bmod n$.
  - Furthermore, given $m_1, m_2, \sigma_1, \sigma_2$, we can let $\sigma^\star = \sigma_1\sigma_2$ and $m^\star = m_1 m_2$.
  - (Have fun: how can I get *arbitrary* signatures from this remark?)

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity
- The message m must be an integer between 2 and $n-1$...
- RSA as described above is not EF-CMA-secure:
  - Take any random value $\sigma^\star$ and compute $m^\star = (\sigma^\star)^e \bmod n$.
  - Furthermore, given $m_1, m_2, \sigma_1, \sigma_2$, we can let $\sigma^\star = \sigma_1\sigma_2$ and $m^\star = m_1m_2$.
  - (Have fun: how can I get *arbitrary* signatures from this remark?)
  - If m is small it is easier to solve...

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity
- The message m must be an integer between 2 and $n-1$...
- RSA as described above is not EF-CMA-secure:
  - Take any random value $\sigma^\star$ and compute $m^\star = (\sigma^\star)^e \bmod n$.
  - Furthermore, given $m_1, m_2, \sigma_1, \sigma_2$, we can let $\sigma^\star = \sigma_1\sigma_2$ and $m^\star = m_1 m_2$.
  - (Have fun: how can I get *arbitrary* signatures from this remark?)
  - If m is small it is easier to solve...

Some of the above issues are handled by using a padding scheme, two popular schemes being RSA-PSS and RSA-OEAP.

Rivest–Shamir–Adleman Signatures

The above signature scheme is widely deployed (credit cards, TLS, etc.). A few remarks are in order:

- We have only computational integrity
- The message m must be an integer between 2 and $n-1$...
- RSA as described above is not EF-CMA-secure:
  - Take any random value $\sigma^\star$ and compute $m^\star = (\sigma^\star)^e \bmod n$.
  - Furthermore, given $m_1, m_2, \sigma_1, \sigma_2$, we can let $\sigma^\star = \sigma_1\sigma_2$ and $m^\star = m_1 m_2$.
  - (Have fun: how can I get *arbitrary* signatures from this remark?)
  - If m is small it is easier to solve...

Some of the above issues are handled by using a padding scheme, two popular schemes being RSA-PSS and RSA-OEAP.

$$\mathrm{OAEP}(m) = (m \oplus G(r)) \| (H(m \oplus G(r)) \oplus r), \quad H \text{ and } G \text{ are "random oracles"}$$

Random what?

The security of digital signatures is a famously difficult problem, that often makes use of one or several random oracles.

An oracle is a black-box entity that can be queried and provides a response according to some prescription.

In the case of an RO, the response are uniformly distributed for new queries.

Random what?

The security of digital signatures is a famously difficult problem, that often makes use of one or several random oracles.

An oracle is a black-box entity that can be queried and provides a response according to some prescription.

In the case of an RO, the response are uniformly distributed for new queries.

ROs are impossible to implement but they provide useful “ideal” models. In practice: hash functions.

Superb so now we have provably secure communications

- ECDHE: key exchange assuming DLOG in the GGM with uniform KDF
- RSA-OAEP: existentially unforgeable signatures assuming FACT and RO access

Under these assumptions we can perform authenticated key exchange that allows us to negociate a session key.

Superb so now we have provably secure communications

- ECDHE: key exchange assuming DLOG in the GGM with uniform KDF
- RSA-OAEP: existentially unforgeable signatures assuming FACT and RO access

Under these assumptions we can perform authenticated key exchange that allows us to negociate a session key.

Great.

Superb so now we have provably secure communications

- ECDHE: key exchange assuming DLOG in the GGM with uniform KDF
- RSA-OAEP: existentially unforgeable signatures assuming FACT and RO access

Under these assumptions we can perform authenticated key exchange that allows us to negociate a session key.

Great.

Or it is?

A few more issues to fix

Why is crypto so complicated?

A few more issues to fix

- What can I use as KDFs and ROs?

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? CRHF

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? CRHF
- How do I know that Alice's signature verification key is *really hers?*

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? CRHF
- How do I know that Alice's signature verification key is *really hers?* PKI

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? CRHF
- How do I know that Alice's signature verification key is *really hers?* PKI
- How do I generate the keys? How do I keep them?

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? — CRHF
- How do I know that Alice's signature verification key is *really hers?* — PKI
- How do I generate the keys? How do I keep them? — HSM

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? CRHF
- How do I know that Alice's signature verification key is *really hers?* PKI
- How do I generate the keys? How do I keep them? HSM
- How do I generate the randomness?

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? CRHF
- How do I know that Alice's signature verification key is *really hers?* PKI
- How do I generate the keys? How do I keep them? HSM
- How do I generate the randomness? TRNG

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? CRHF
- How do I know that Alice's signature verification key is *really hers?* PKI
- How do I generate the keys? How do I keep them? HSM
- How do I generate the randomness? TRNG
- How fast is it?

A few more issues to fix

Why is crypto so complicated?

- What can I use as KDFs and ROs? CRHF
- How do I know that Alice's signature verification key is *really hers?* PKI
- How do I generate the keys? How do I keep them? HSM
- How do I generate the randomness? TRNG
- How fast is it? Slow.

Hybrid PK-SK

Idea:

Hybrid PK-SK

Idea: once session key is negociated, use faster, symmetric algorithms.

Hybrid PK-SK

Idea: once session key is negociated, use faster, symmetric algorithms.

- Symmetric encryption (e.g. Chacha20, AES) instead of public-key encryption
- MACs (e.g. Poly1305) instead of public-key signatures
- A mode of operation must be chosen.

Note that in general, the security of such constructions is not as clear as that of public-key crypto though.

Hybrid PK-SK

Idea: once session key is negociated, use faster, symmetric algorithms.

- Symmetric encryption (e.g. Chacha20, AES) instead of public-key encryption
- MACs (e.g. Poly1305) instead of public-key signatures
- A mode of operation must be chosen.

Note that in general, the security of such constructions is not as clear as that of public-key crypto though.

Go to a HTTPS website and look at what they're using!

1 slide summary

- PKI certification $\rightarrow$ long term, personal signature key
- Authenticated key exchange $\rightarrow$ ephemeral, shared session key
- Symmetric encryption+MAC (or AEAD) $\rightarrow$ confid + integr. on channel
- Key sizes chosen to ensure heuristic effort of about 2^{128} $\rightarrow$ security level
- Key management is essential and critical Use HSMs
- Randomness is essential and critical Use TRNGs

E.g.: Use Curve25519 + EdDSA + SHA3 + AES-GCM-SIV

Take away

- Learn about cryptography (I can give references)
- It is very easy to design a system that the designers themselves can't break, especially if the designers have little experience in breaking things. (Schneier's Law)
- Proofs have preconditions and precise scope. GAME-MEANS notation.
- Using proper cryptography is necessary but not sufficient for security

Pause

After the pause: Internet Lab!

Bonus: flipping a coin over a telephone

Note f is a one-way function, $\|$ denotes concatenation.

1. Alice chooses random numbers a_0, a_1 and computes $b_i = f(i\|a_i)$.
2. Alice sends Bob b_0 and b_1, and indicates that she bets on b_A, $A \in \{0, 1\}$.
3. Bob flips a coin and announces the result B.
4. Alice reveals A and a_0, a_1.
5. Bob can check that $f(A\|a_A) = b_A$, and if $A = B$ then Alice wins otherwise she looses.