Cryptography

Key Exchange & Digital Signatures

University of Birmingham

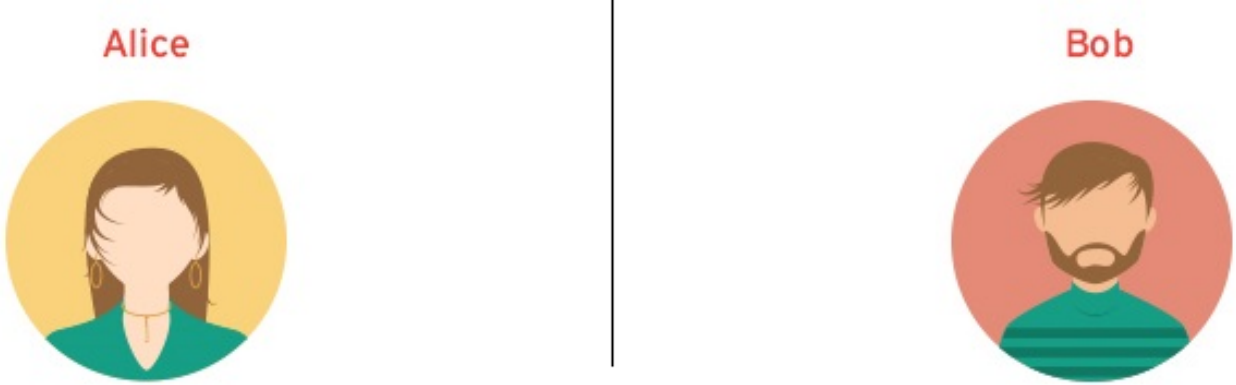
Autumn Term 2017

Lecturer: David Galindo

UNIVERSITY OF BIRMINGHAM | Security and Privacy

Diffie-Hellman Key Exchange

- Enables Alice and Bob to **establish a shared secret key** that nobody else can compute, without having talked to each other before



Key Exchange

- Enables Alice and Bob to **establish a shared secret key** that nobody else can compute, without having talked to each other before
- Key generation
  - Let p and $q|p-1$ be prime integers, and let g be a generator of $G_q \subset \mathbb{Z}_p^\star$
  - Let p, q and g be public
  - Alice generates a random $a \in \mathbb{Z}_q$ and publishes $A = g^a \mod p$. She keeps a secret
  - Bob generates a random $b \in \mathbb{Z}_q$ and publishes $B = g^b \mod p$. He keeps b secret

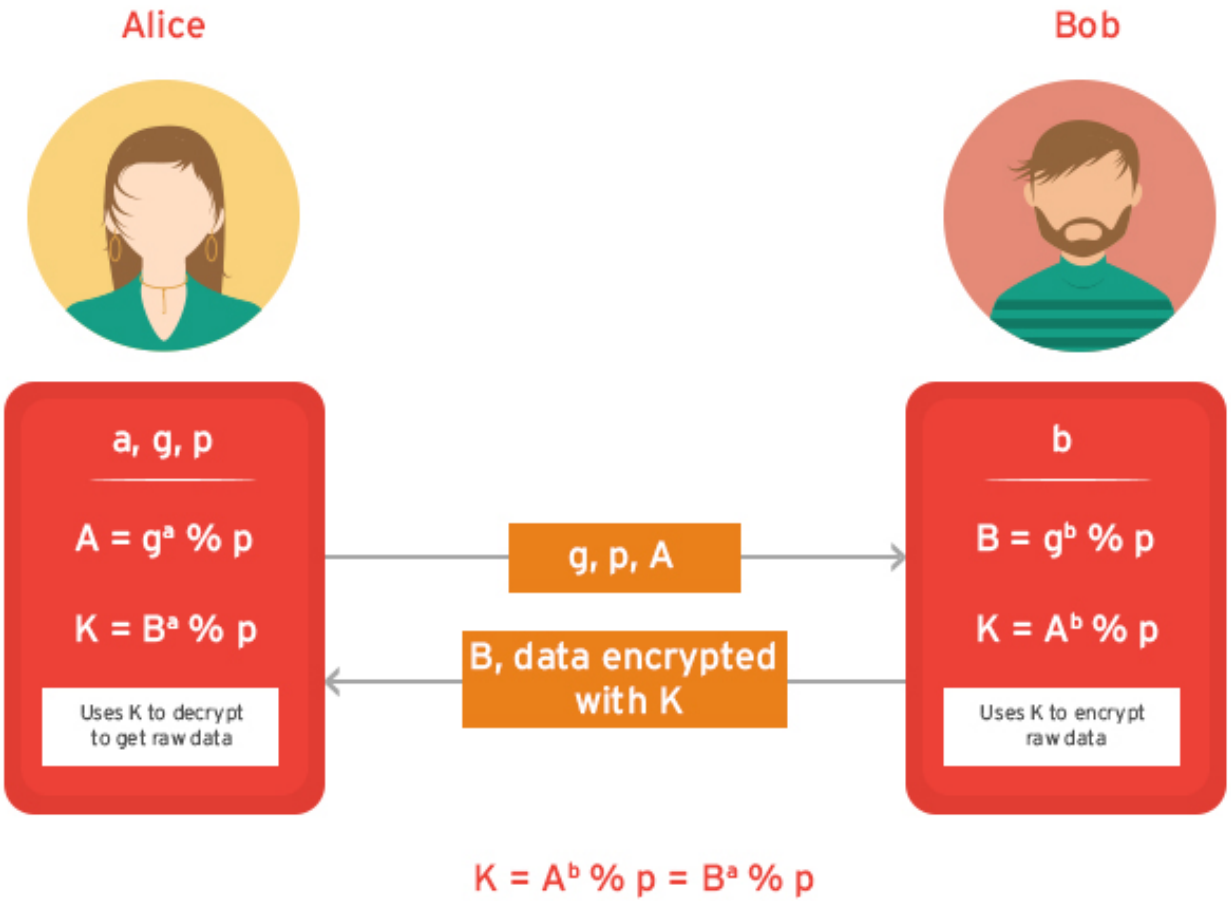
Diffie-Hellman protocol (1976)

- Key establishment
  - Alice sends A to Bob
    - Bob ends up knowing $A = g^a, b$
  - Bob sends B to Alice
    - Alice ends up knowing $B = g^b, a$
  - Alice computes $K_a = B^a \mod p$
  - Bob computes $K_b = A^b \mod p$

$$K_a = B^a = (g^b)^a = g^{ab} = (g^a)^b = A^b = K_b \mod p$$

- Alice and Bob now share the same key $K = K_a = K_b$
  - Without knowing a or b, adversary is unable to compute K
  - Computing $g^{ab} \mod p$ from $g^a \mod p$ and $g^b \mod p$ is equivalent to solving the *Diffie-Hellman problem*, for which no efficient algorithm is known

Diffie-Hellman Key Exchange in practice



$K = A^b \% p = B^a \% p$

Man-in-the-Middle attacks

Alice and Bob want to exchange data confidentially, w/o Mallory learning the content

	Alice		Mallory		Bob
Step 1.	a	$\longrightarrow$	g^a		
Step 2.	g^m	$\longleftarrow$	m		
Step 3.	g^{ma}		g^{ma}		
Step 4.			n	$\longrightarrow$	g^n
Step 5.			g^b	$\longleftarrow$	b
Step 6.			g^{nb}		g^{nb}
End Result	g^{ma}	$\longleftrightarrow$	g^{ma}, g^{nb}	$\longleftrightarrow$	g^{nb}

Alice thinks she is talking to Bob (w/o Mallory learning the content)
Bob thinks she is talking to Alice (w/o Mallory learning the content)
But they are actually both talking to Mallory!

Authenticating Public Keys

One way to stop MiTM attacks is by guaranteeing authenticity of g^a, g^b

Can achieve this with public-key cryptography by using **digital signatures**!

A Certification Authority (CA) is used to **authenticate public keys**, i.e. should give the relation between the public key and the identity of his owner

The CA **signs the relationship** under the widely known public key of the certification authority vk_{CA}

Certificate Process

Alice generates (pk, sk) and sends pk to CA

CA performs identity check

Alice proves knowledge of secret key sk to CA

CA issues certificate to Alice

Alice sends certificate to Bob

Bob verifies certificate and extracts Alice's pk

Certificate Process

- Proof of Knowledge
  - The CA might have Alice decrypt a ciphertext under $K = g^{sk \cdot \hat{sk}}$ for a CA's temporary public key $\hat{pk}$ to ensure that Alice knows the corresponding secret key sk
  - This ensures Alice has not copied someone else's key

Certificate Process

- Proof of Knowledge
  - The CA might have Alice decrypt a ciphertext under $K = g^{sk \cdot \hat{sk}}$ for a CA's temporary public key $\hat{pk}$ to ensure that Alice knows the corresponding secret key sk
  - This ensures Alice has not copied someone else's key
  - What if the pair (pk, sk) is used by Alice for a digital signature scheme?

Certificate Issuance

Once the CA is convinced that pk belongs to Alice it forms a certificate

$$CERT_A = (CERTDATA, \sigma),$$

where σ is the CA's signature on $CERTDATA$, computed under the CA's secret key sk_{CA}

$CERTDATA$ consists of:

- pk, ID_{Alice}
- Name of CA
- Expiry date of certificate
- Restrictions
- Security level
- ...

The certificate $CERT_A$ is returned to Alice

Certificate Usage

Alice can send $CERT_A$ to Bob who will:

- Parse $CERT_A$ as $(CERTDATA, \sigma)$
- Check $\text{Verify}(pk_{CA}, CERTDATA, \sigma) = 1$ where pk_{CA} is CA's public key
- Parse $CERTDATA$ as $(pk, ID_{Alice}, \text{expiry}, \ldots)$
- Check certificate has not expired
- $\ldots$

If all is well we are ready for usage

How does Bob get pk_{CA}

- CA public keys are embedded in software such as your browser

Digital Signature Schemes

- Relate an individual, through a **digital string**, to a document
- Would like to achieve all features of hand written signatures... plus more
- Should be able to argue that **forgery is difficult** based on hard computational problem

Hand Written Signatures

- Relate an individual, through a **handwritten** signature, to a document
- A signature can be **verified** against a prior authenticated one, which was signed in person in a bank, or in the presence of notary public
- Should **hard to forge**
- Are **legally binding** (convince a third party)

Some terminology

A **wet signature** is created when a person *physically* marks a document. The word *wet* implies that the signature requires time to dry (e.g. it was made with ink)

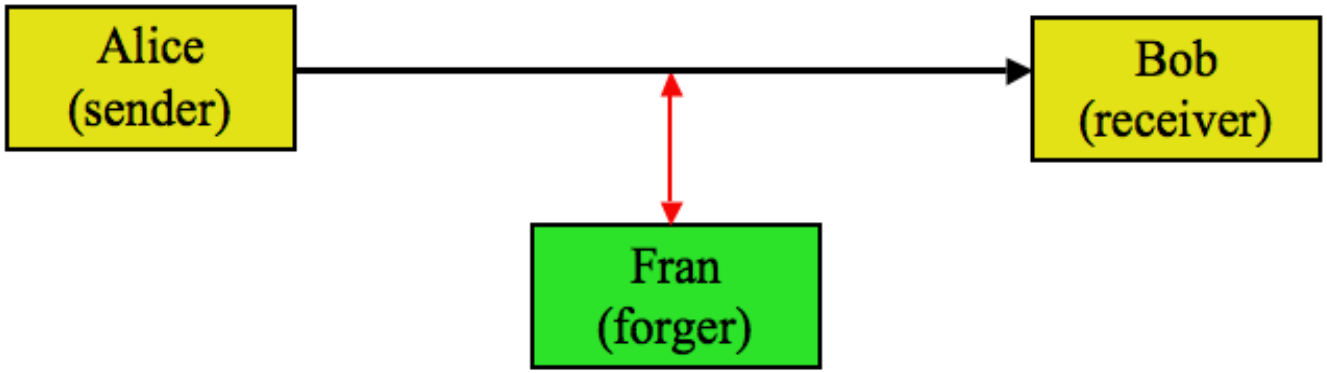
The term **electronic signature** refers to the acknowledgement or adoption of an *electronic* message, transaction or document. Examples include:

- An image of a handwritten signature on an electronic document
- A (PIN) entered into a bank ATM
- Clicking *agree* or *disagree* on an electronic terms and agreements contract
- A handwritten but digitally captured signature made on a touch device

A **digital signature** is a *cryptographic* signature

Example 1: Software Updates

- Alice (e.g. Microsoft) sells a piece of software
- Later, Alice periodically sends updates to Bob1, Bob2 (its customers) and signs it

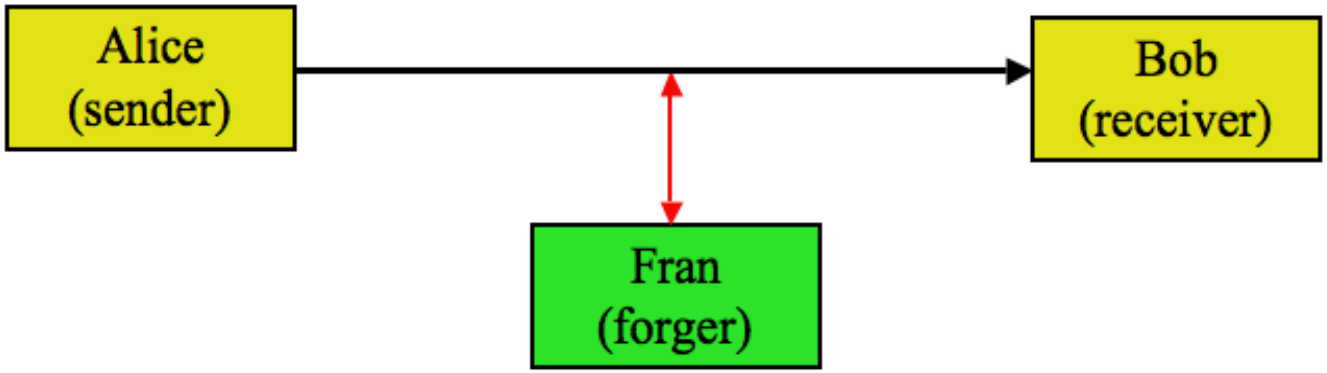


- Fran should not be able to generate a **fake update** on behalf of Alice

Question: Why is it **not advisable** to use MAC?

Example 1: Software Updates

- Alice (e.g. Microsoft) sells a piece of software
- Later, Alice periodically sends updates to Bob1, Bob2 (its customers) and signs it



- Fran should not be able to generate a **fake update** on behalf of Alice

Question: Why is it **not advisable** to use MAC?

Inefficient! Many customers, not usable to share a MAC key with every user! But many others...

Example 2: Signing a Check

Alice digitally **signs a check** of $10. We need that:

- Bob can't cheat and change the check value to $10000!
- Alice can't cheat and claim it is not her check
- Everyone should be able to verify the signature

Question: Why **can't we** use a MAC with under a shared symmetric key to achieve that?

Example 2: Signing a Check

Alice digitally **signs a check** of $10. We need that:

- Bob can't cheat and change the check value to $10000!
- Alice can't cheat and claim it is not her check
- Everyone should be able to verify the signature

Question: Why **can't we** use a MAC with under a shared symmetric key to achieve that?

- MAC protects Alice and Bob against cheating by a 3rd party
- MAC does not protect them against **cheating each other**
- Signatures are **publicly verifiable, transferable** and **non-repudiable**

These are **qualitative advantages** of digital signatures over MACs

Syntax of Digital Signatures

A digital signature $\mathcal{S}$ is the asymmetric cryptography analog of a MAC. It consists of three algorithms $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Verify})$:

- $\mathsf{KG}(\lambda)$ on input a security parameter λ outputs a pair of verification/signing keys (vk, sk), whereby vk is public, and sk is secret to the signer
- $\mathsf{Sign}(sk, m; r)$ on inputs a verification key vk, string m and (eventually) randomness r outputs a signature σ
- $\mathsf{Verify}(vk, m, \sigma)$ on inputs a verification key vk, a string m and a signature σ, outputs yes/no, whereby yes means that σ is a valid signature on the digital document m

Usage

Step 1. Signer runs $(vk, sk) \leftarrow \mathrm{KG}(\lambda)$ and stores sk secretly

Step 2. Signer publicly announces vk

Step 3. Verifier accepts vk

Step 4. Signer produces a signature σ on a given document M given the signing key sk

Step 5. Any third-party who had previous knowledge of vk can verify the signature by running $\mathrm{Verify}(vk, M, \sigma)$

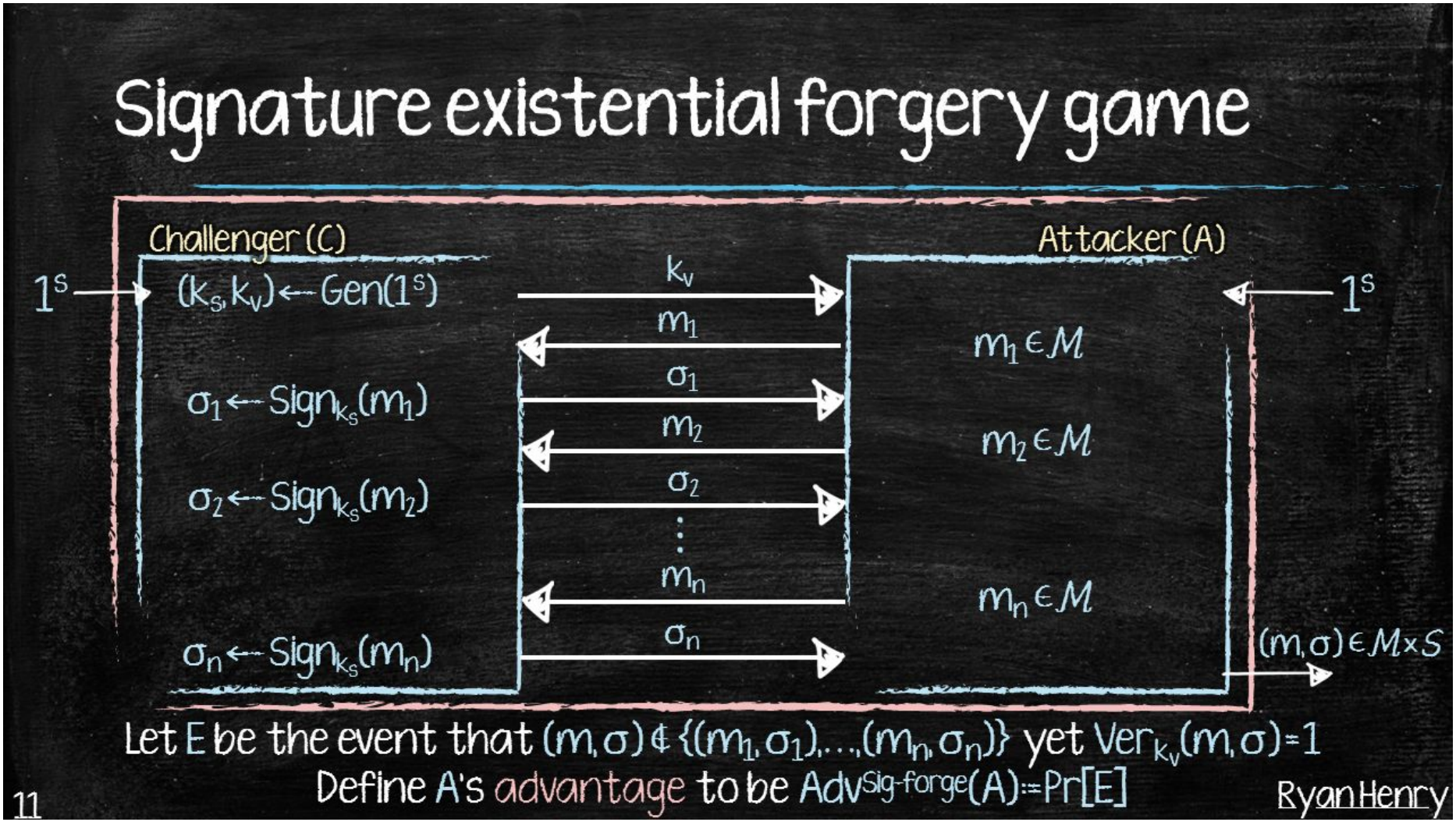
Unforgeability of a signature scheme

Definition

The *unforgeability game* between Challenger and Attacker is defined as follows:

- **Init.** Challenger runs $(vk, sk) \leftarrow \mathrm{KG}(\lambda)$ and passes vk to the attacker
- **Find.** Attacker does some computations and may ask challenger to sign messages $m_1, \ldots, m_n$
- Challenger responds with signatures $s_1, \ldots, s_n$
- **End.** The attacker outputs a pair (m, s)

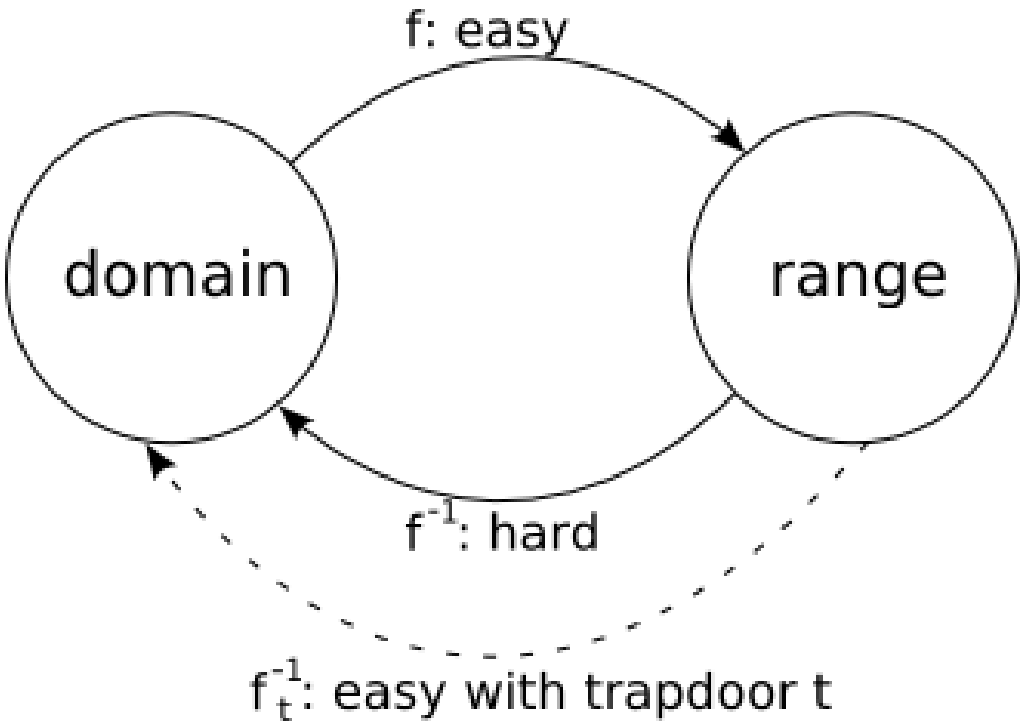
The attacker wins the unforgeability game if (m, s) is a valid signature for m, and m did not appear in the **Find** phase.


Signature existential forgery game
Challenger (C)
Attacker (A)
1^s
$(k_s, k_v) \leftarrow Gen(1^s)$
k_v
m_1
$m_1 \in M$
$\sigma_1 \leftarrow Sign_{k_s}(m_1)$
σ_1
m_2
$m_2 \in M$
$\sigma_2 \leftarrow Sign_{k_s}(m_2)$
σ_2
m_n
$m_n \in M$
$\sigma_n \leftarrow Sign_{k_s}(m_n)$
σ_n
$(m, \sigma) \in M \times S$
Let E be the event that $(m, \sigma) \notin \{(m_1, \sigma_1), \ldots, (m_n, \sigma_n)\}$ yet $Ver_{k_v}(m, \sigma) = 1$
Define A's advantage to be $Adv^{Sig\text{-}forge}(A) := Pr[E]$
11
Ryan Henry

Definition

We call a digital signature scheme **secure against existential forgery** (or *unforgeable*) if any attacker has only a negligible chance of winning the unforgeability game

Basic Ingredient: Trapdoor One-Way Functions



www.cs.cornell.edu

Signature from Trapdoor One-Way Bijections

For **digital signatures** we can build from a function:

- is **easy to compute** given the public key PK
  - i.e. given PK, x anyone can compute $y = f_{PK}(x)$
- invertible efficiently on knowledge of the **trapdoor** SK
  - i.e. given $SK, y = f_{PK}(x)$ one can compute $f_{SK}^{-1}(y) = x$
- **infeasible to invert** w/o knowledge of SK
  - i.e. given $PK, y = f_{PK}(x)$ no one can compute $f_{SK}^{-1}(f_{PK}(x))$

Signature from Trapdoor One-Way Bijections

For **digital signatures** we can build from a function:

- is **easy to compute** given the public key PK
  - i.e. given PK, x anyone can compute $y = f_{PK}(x)$
- invertible efficiently on knowledge of the **trapdoor** SK
  - i.e. given $SK, y = f_{PK}(x)$ one can compute $f_{SK}^{-1}(y) = x$
- **infeasible to invert** w/o knowledge of SK
  - i.e. given $PK, y = f_{PK}(x)$ no one can compute $f_{SK}^{-1}(f_{PK}(x))$

How to define x as a function of the message m to be signed?
How to define the signature σ?

Signature from Trapdoor One-Way Bijections

For **digital signatures** we can build from a function:

- is **easy to compute** given the public key PK
  - i.e. given PK, x anyone can compute $y = f_{PK}(x)$
- invertible efficiently on knowledge of the **trapdoor** SK
  - i.e. given $SK, y = f_{PK}(x)$ one can compute $f_{SK}^{-1}(y) = x$
- **infeasible to invert** w/o knowledge of SK
  - i.e. given $PK, y = f_{PK}(x)$ no one can compute $f_{SK}^{-1}(f_{PK}(x))$

How to define x as a function of the message m to be signed?
How to define the signature σ?

What about setting $\sigma = f_{SK}^{-1}(m)$ and verifying $m = f_{PK}(f_{SK}^{-1}(m))$?

RSA-Full Domain Hash

Let $F_{N,e} : \mathbb{Z}_N^\star \to \mathbb{Z}_N^\star$ be $F_{N,e}(x) = x^e \mod N$ be the **RSA Trapdoor One-Way Bijection**, and $F_d^{-1} : \mathbb{Z}_N^\star \to \mathbb{Z}_N^\star$ with $F_d^{-1}(y) = y^d \mod N$ be the inverse function.

Let $H : \{0,1\}^\star \to \mathbb{Z}_N^\star$ be a secure hash function

- Sign$((N, e, d), m)$: to sign $m \in \{0,1\}^\star$, compute $\sigma = F_d^{-1}(H(m)) = H(m)^d \mod N$
- Verify$((N, e), m, \sigma)$: Given $m \in \{0,1\}^\star$ and $\sigma \in \mathbb{Z}_N^\star$, output yes/no depending on whether $H(m) = F_{N,e}(\sigma) = \sigma^e \mod N$

RSA-Full Domain Hash

Let $F_{N,e} : \mathbb{Z}_N^\star \to \mathbb{Z}_N^\star$ be $F_{N,e}(x) = x^e \mod N$ be the **RSA Trapdoor One-Way Bijection**, and $F_d^{-1} : \mathbb{Z}_N^\star \to \mathbb{Z}_N^\star$ with $F_d^{-1}(y) = y^d \mod N$ be the inverse function.

Let $H : \{0,1\}^\star \to \mathbb{Z}_N^\star$ be a secure hash function

- Sign$((N, e, d), m)$: to sign $m \in \{0,1\}^\star$, compute $\sigma = F_d^{-1}(H(m)) = H(m)^d \mod N$
- Verify$((N, e), m, \sigma)$: Given $m \in \{0,1\}^\star$ and $\sigma \in \mathbb{Z}_N^\star$, output yes/no depending on whether $H(m) = F_{N,e}(\sigma) = \sigma^e \mod N$

Fact

RSA-FDH is an unforgeable signature scheme under that assumption that the RSA problem is infeasible to break

What we need from a secure H

Suppose an adversary can “invert" hash function H on input a random $y \in \mathbb{Z}_N^\star$, meaning finding a string M such that $H(M) = y$

Then a forgery is easy:

- Attacker chooses σ randomly in $\mathbb{Z}_N^\star$
- Attacker computes $\sigma^e \mod N$
- Attacker computes M such that $H(M) = \sigma^e \mod N$
- Attacker outputs the pair (M, σ) as the forgery

Therefore, H needs to be **one-way**

What we need from H

Suppose an adversary can find collisions from H, meaning finding $M_1 \neq M_2$ such that $H(M_1) = H(M_2)$

Then $H(M_1)^d = H(M_2)^d \mod N$ and a forgery is easy:

- Attacker requests a signature σ on M_1
- Attacker outputs the pair (M_2, σ) as the forgery

Therefore H needs to be **collision-resistant**

What we need from H

Suppose H has **structural properties**, such as

$$H(M_1 \oplus M_2) = H(M_1) \cdot H(M_2)$$

for any M_1, M_2 of the same bit-size

Then a forgery is again easy:

- Attacker requests a signature σ_1 on M_1
- Attacker requests a signature σ_2 on M_2
- Attacker sets $\sigma = \sigma_1 \cdot \sigma_2$ and $M = M_1 \oplus M_2$
- Attacker outputs the pair (M, σ) as the forgery

H as a Random oracle

We know H should be one-way, collision-resistant and not have structural properties

Which properties are needed to guarantee that the RSA-FDH signature scheme is *unforgeable*?

We ask that H should "behave" as a *Random Oracle*:

- $H : X \to Y$ "behaves" as a public random function, namely:
  - $H(M_1) = H(M_2)$ with probability $1/|Y|$ for any $M_1 \neq M_2$
- Thus, an attacker regards H as a lookup table (i.e. it does not exploit the inner workings of H)

The Random Oracle Model heuristic

In practice, instantiate the function H with a hash function and hope that security still holds

The rationale behind the Random Oracle approach is that if the protocol becomes insecure, then we would find an aspect in which the hash function does not behave like a random function

It is an useful intermediary step between **ad-hoc constructions** with high efficiency **but no proof** of security and **theoretical inefficient** solutions which are **provably-secure**

Instantiate H in RSA-FDH

We need to build $H : \{0,1\}\star \to \mathbb{Z}_N^\star$, and in practice, this boils down to $H : \{0,1\}^\star \to \{0,1\}^{2048}$

Existing hash functions have output lengths 160, 224, 256, 384, 512 bits (i.e. SHA1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384)

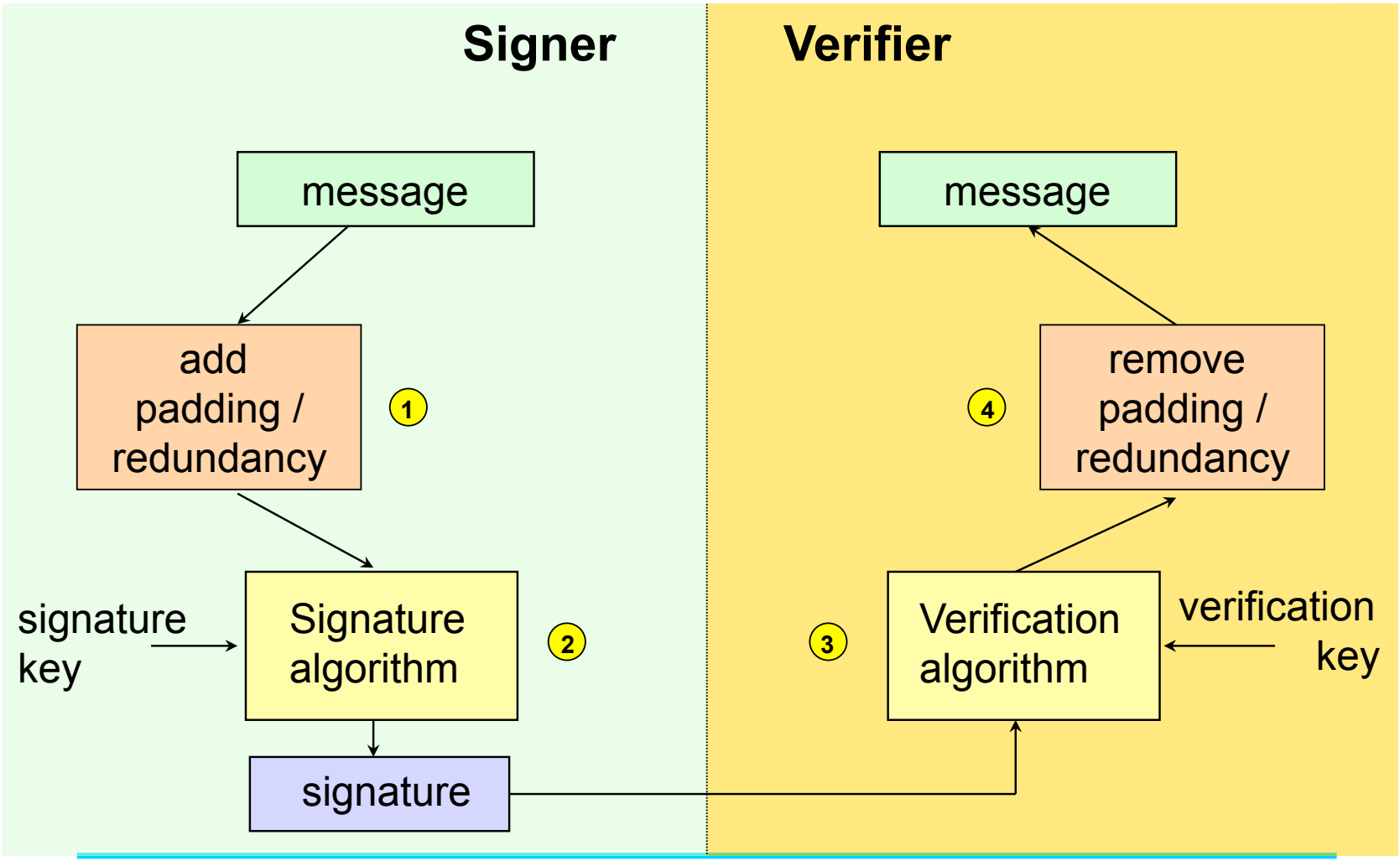
A sound approach is to define

$$H(M) = \text{SHA-512}(1||M)\Big|\Big|\text{SHA-512}(2||M)\Big|\Big|\text{SHA-512}(3||M)\Big|\Big|\text{SHA-512}(4||M)$$

or

$$H(M) = \text{SHA-256}(1||M)\Big|\Big|\ldots\Big|\Big|\text{SHA-256}(8||M)$$

RSA signatures with message recovery



PSS [BR96]

Signer $pk = (N, e)$ and $sk = (N, d)$

algorithm $\mathcal{S}^{h,g_1,g_2}_{N,d}(M)$	algorithm $\mathcal{V}^{h,g_1,g_2}_{N,e}(M, x)$
$r \xleftarrow{\$} \{0,1\}^{160}$	$y \leftarrow x^e \bmod N$
$w \leftarrow h(M \,\|\|\, r)$	$b \,\|\|\, w \,\|\|\, r^* \,\|\|\, P \leftarrow y$
$r^* \leftarrow g_1(w) \oplus r$	$r \leftarrow r^* \oplus g_1(w)$
$y \leftarrow 0 \,\|\|\, w \,\|\|\, r^* \,\|\|\, g_2(w)$	if $(g_2(w) \neq P)$ then return 0
return $y^d \bmod N$	if $(b = 1)$ then return 0
	if $(h(M \,\|\|\, r) \neq w)$ then return 0
	return 1

Here h, g_1: $\{0,1\}^* \to \{0,1\}^{160}$ and g_2: $\{0,1\}^* \to \{0,1\}^{k-321}$ are random oracles where $k = |N|$.

Probabilistic Signature Signing (PSS) scheme - usage

- RSA PKCS#1 v2.1.
- IEEE P1363a
- ANSI X9.31
- RFC 3447
- ISO/IEC 9796-2
- CRYPTREC
- NESSIE

Schnorr signature scheme (1991)

Not standardized yet, as it was patented (expired, 2008)

KG(λ)

- choose a 2048 bit prime p such that:
  - a 256 bit prime q, such that q divides $p-1$ and q is the order of a subgroup $G_q = \langle g \rangle$ of $\mathbb{Z}_p^*$
- Choose $H : \{0,1\}^\star \to \mathbb{Z}_q$ a hash function
- Choose a random x from $\{0, \ldots, q-1\}$ (i.e. from $\mathbb{Z}_q$)
- Compute $y = g^x \bmod p$
- Publish the public key $vk = (p, q, g, y, H)$
- Retain the private key $sk = x$

Schnorr signature scheme (1991)

Sign(sk, M) to sign a bit-string M do:

- Choose a random r from $\{0, \ldots, q-1\}$
- Compute $s = H(M \| g^r)$
- Compute $t = (r + x \cdot s) \bmod q$
- Output signature $\sigma = (s, t)$

Verify(vk, σ, m) works as follows:

- Parse σ as (s, t)
- Accept the signature if $H(M \| g^t y^{-s}) = s$
- Otherwise reject the signature

It can be seen that Verify$\big(vk, \text{Sign}(sk, m), m\big)$ outputs accept if $(vk, sk) \leftarrow \text{KG}(\lambda)$

Fact

The Schnorr signature scheme is unforgeable under the Discrete Logarithm assumption in the Random Oracle Model

DSA (Digital Signature Algorithm) - 1991

DSA is adopted in standard FIPS 186-1 to FIPS 186-4

KG(λ)

- choose a 2048 bit prime p such that:
  - a 256 bit prime q, such that q divides $p-1$ and q is the order of a subgroup $G_q = \langle g \rangle$ of $\mathbb{Z}_p^*$
- the cryptographic hash function $H : \{0,1\}^\star \rightarrow \{0,1\}^{256}$
- Choose a random x from $\mathbb{Z}_q$
- Compute $y = g^x \bmod p$
- Publish the public key $vk = (p, q, g, y, H)$
- Retain the private key $sk = x$

DSA (Digital Signature Algorithm) - 1991

$\text{Sign}(sk, M)$ to sign a bit-string M do:

- Choose a random r from $\mathbb{Z}_q$
- Compute $s = (g^r \bmod p) \bmod q$
- Compute $t = (H(M) + x \cdot s) \cdot r^{-1} \bmod q$
- Output the signature $\sigma = (s, t)$

$\text{Verify}(vk, \sigma, m)$ works as follows:

- Calculate $u_1 = H(M) \cdot t^{-1} \bmod q$
- Calculate $u_2 = s \cdot t^{-1} \bmod q$
- Accept the signature if $(g^{u_1} \cdot y^{u_2}) \bmod p \bmod q = s$
- Otherwise reject the signature

It can be seen that $\text{Verify}\big(vk, \text{Sign}(sk, m), m\big)$ outputs accept if $(vk, sk) \leftarrow \text{KG}(\lambda)$

DSA (Digital Signature Algorithm) - 1991

It can be seen that $\text{Verify}\big(vk, M, \text{Sign}(sk, M)\big)$ outputs accept if $(vk, sk) \leftarrow \text{KG}(\lambda)$

Let $\sigma := \text{Sign}(sk, M)$. Thus $\sigma = (s, t)$ is such that $t = (H(M) + x \cdot s) \cdot r^{-1} \bmod q$ and $s = (g^r \bmod p) \bmod q$. Then

$$
\begin{aligned}
&(g^{u_1} \cdot y^{u_2}) \bmod p = \\
&g^{H(M)\cdot t^{-1}} \cdot g^{xst^{-1}} \bmod p = \\
&g^{t^{-1}(H(M)+xs)} \bmod p = \\
&g^r \bmod p
\end{aligned}
$$

Finally, $(g^{u_1} \cdot y^{u_2}) \bmod p \bmod q = (g^r \bmod p) \bmod q = s$

Useful references

- Handout 10 (in Canvas)
- Introduction to Modern Cryptography by Benny Applebaum (TAU, IL)

  http://tau-crypto12.wikidot.com
- Introduction to Modern Cryptography by Mihir Bellare (UCSD, USA)

  https://cseweb.ucsd.edu/~mihir/cse107/index.html