HYBRID ENCRYPTION
and DIGITAL SIGNATURES

Recall: Hybrid encryption

Given:

- A KEM $\mathcal{KE} = (\mathcal{KK}, \mathcal{EK}, \mathcal{DK})$ with key length k
- A symmetric encryption scheme $\mathcal{SE} = (\mathcal{KS}, \mathcal{ES}, \mathcal{DS})$ for which $\mathcal{KS}$ returns random k-bit keys.

Hybrid encryption associates to the above a PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ via:

Alg $\mathcal{K}$	**Alg** $\mathcal{E}_{ek}(M)$	**Alg** $\mathcal{D}_{dk}((C_a, C_s))$
$(ek, dk) \xleftarrow{\$} \mathcal{KK}$	$(K, C_a) \xleftarrow{\$} \mathcal{EK}_{ek}$	$K \leftarrow \mathcal{DK}_{dk}(C_a)$
Return (ek, dk)	$C_s \xleftarrow{\$} \mathcal{ES}_K(M)$	$M \leftarrow \mathcal{DS}_K(C_s)$
	Return (C_a, C_s)	Return M

Above, it is understood that if any input to an algorithm is $\perp$, then so is the output.

Where we are

We know how to achieve IND-ATK-secure PKE given

- An IND-ATK-secure KEM, and
- An IND-ATK-secure symmetric encryption scheme

We have plenty of symmetric encryption schemes:

- For the IND-CPA case: AES-CTR$, AES-CBC$, ...
- For the IND-CCA case: Encrypt-then-Mac, OCB, GCM, ...

But simpler, deterministic choices are possible too, since security is only required against adversaries B_s making 1 **LR** query.

We need KEMs.

We will build KEMs using number theory, considering in turn using the DL problem and using RSA.

Hashing in KEMs

Our KEMs may use (public, keyless) functions $\mathbf{H}_i$: $\{0,1\}^* \to \{0,1\}^{\ell_i}$, for $1 \leq i \leq n$.

The number n of them, and their output lengths, depend on the scheme. Usually $n = 1$ or $n = 2$.

In practice (implementation), these are built from cryptographic hash functions as discussed next.

Proofs of security for the KEMs use the Random Oracle Model (ROM) [BR93] in which $\mathbf{H}_1, \ldots, \mathbf{H}_n$ are modeled as independent random functions. They are formalized as game procedures to which scheme algorithms, *as well as the adversary*, have oracle access, and are thus called Random Oracles (ROs).

Practical choices for the $\mathbf{H}_i$

We seek suitable functions $\mathbf{H}_i$: $\{0,1\}^* \to \{0,1\}^{\ell_i}$, for $1 \le i \le n$.

$\mathrm{SHAKE256}$ is an XOF (eXtendable Output length Function) that takes an input indicating the number of output bits returned.

So we could set $\mathbf{H}_i(x) = \mathrm{SHAKE256}(\langle i \rangle \| x, \ell_i)$ where $\langle i \rangle$ is a 1-byte encoding of i and we assume $n < 2^8$.

We could also set $\mathbf{H}_i(x)$ to the first ℓ_i bits of the sequence

$$\mathrm{SHA256}(\langle 0 \rangle \| \langle i \rangle \| x) \,\|\, \mathrm{SHA256}(\langle 1 \rangle \| \langle i \rangle \| x) \,\|\, \cdots \,\|\, \mathrm{SHA256}(\langle 2^8 - 1 \rangle \| \langle i \rangle \| x)$$

This assumes $\ell_i \le 2^8 \cdot 256$.

Heuristically, we desire that $\mathbf{H}_1, \ldots, \mathbf{H}_n$ "behave like independent random functions." But there is no corresponding formal definition.

KEMs from DL?

Let $G = \langle g \rangle$ be a cyclic group of order m in which the DL problem is hard. Can we design a KEM $\mathcal{KE} = (\mathcal{KK}, \mathcal{EK}, \mathcal{DK})$ whose IND-CPA security reduces to DL?

How about: Let the receiver's encryption key be g. Let $\mathcal{EK}_g$ pick $x \xleftarrow{\$} \mathbf{Z}_m$ and return (x, X) where $X = g^x$.

Then obtaining x from X requires solving DL, and would be hard for an adversary.

So are we done?

KEMs from DL?

Let $G = \langle g \rangle$ be a cyclic group of order m in which the DL problem is hard. Can we design a KEM $\mathcal{KE} = (\mathcal{KK}, \mathcal{EK}, \mathcal{DK})$ whose IND-CPA security reduces to DL?

How about: Let the receiver's encryption key be g. Let $\mathcal{EK}_g$ pick $x \xleftarrow{\$} \mathbf{Z}_m$ and return (x, X) where $X = g^x$.

Then obtaining x from X requires solving DL, and would be hard for an adversary.

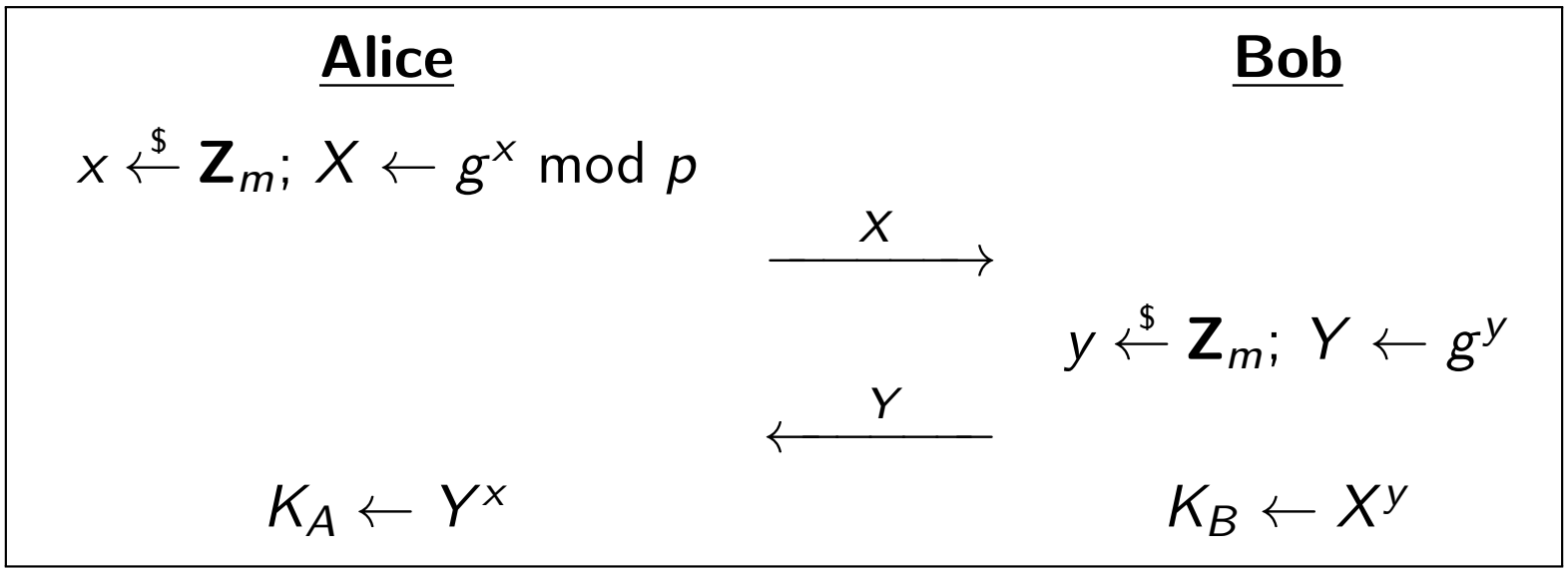
So are we done?

No. The legitimate receiver has no way to decrypt X, to obtain x, short of computing DL.

A sign that something is amiss is that, in the above scheme, the receiver has no decryption key.

Recall DH Secret Key Exchange

Let $G = \langle g \rangle$ be a cyclic group of order m.



- $Y^x = (g^y)^x = g^{xy} = (g^x)^y = X^y$, so $K_A = K_B$
- Adversary is faced with the CDH problem, which needs to be assumed hard for security. This is a stronger requirement than hardness of DL.

From key exchange to PKE

We can turn DH key exchange into a PKE scheme via

- Alice has encryption key $X = g^x$ and decryption key $x \xleftarrow{\$} \mathbf{Z}_{p-1}$
- If Bob wants to encrypt message M for Alice, he
  - Picks $y \xleftarrow{\$} \mathbf{Z}_{p-1}$ and sends $Y = g^y$ to Alice
  - Computes $Z = (g^x)^y = g^{xy}$, hashes it to get a key K, encrypts M symmetrically under K to get a ciphertext C_s, and sends C_s to Alice.
- Alice can recompute $Z = Y^x = g^{xy}$ using her decryption key x. Then she can recompute K and decrypt C_s under it to get M.

The adversary is faced with either solving CDH or breaking the symmetric encryption scheme.

The DHIES scheme

Let $G = \langle g \rangle$ be a cyclic group of order m and H: $G \to \{0,1\}^k$ a (public) hash function. The DHIES PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined for messages $M \in \{0,1\}^k$ via

Alg $\mathcal{K}$	**Alg** $\mathcal{E}_X(M)$	**Alg** $\mathcal{D}_x(Y, W)$
$x \xleftarrow{\$} \mathbf{Z}_m$	$y \xleftarrow{\$} \mathbf{Z}_m$; $Y \leftarrow g^y$	$K \leftarrow Y^x$
$X \leftarrow g^x$	$K \leftarrow X^y$	$M \leftarrow H(K) \oplus W$
return (X, x)	$W \leftarrow H(K) \oplus M$	return M
	return (Y, W)	

Correct decryption is assured because $K = X^y = g^{xy} = Y^x$

Note: This is a simplified version of the actual scheme.

Security of DHIES

The DHIES scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ associated to cyclic group $G = \langle g \rangle$ and (public) hash function H can be proven IND-CPA assuming

- CDH is hard in G, and
- H is a "random oracle," meaning a "perfect" hash function.

In practice, $H(K)$ could be the first k bits of the sequence

$$\text{SHA256}(0^8 \| K) \| \text{SHA256}(0^7 1 \| K) \| \cdots$$

ECIES

ECIES is DHIES with the group being an elliptic curve group.

ECIES features:

Operation	**Cost**
encryption	2 160-bit exp
decryption	1 160-bit exp
ciphertext expansion	160 bits

ciphertext expansion = (length of ciphertext) - (length of plaintext)

Plain-RSA PKE scheme

Let $\mathcal{K}_{\text{rsa}}$ be an RSA generator. The plain RSA PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined via:

Alg $\mathcal{K}$	**Alg** $\mathcal{E}_{(N,e)}(M)$	**Alg** $\mathcal{D}_{(N,d)}(C)$
$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$ Return $((N, e), (N, d))$	$C \leftarrow M^e \bmod N$ return C	$M \leftarrow C^d \bmod N$ return M

Above, (N, e) is the encryption key and (N, d) is the decryption key.

Decryption correctness: The "easy-backwards with trapdoor" property implies that for all $M \in \mathbf{Z}_N^*$ we have $\mathcal{D}_{dk}(\mathcal{E}_{ek}(M)) = M$.

Note: The message space is $\mathbf{Z}_N^*$. Messages are assumed to be all encoded as strings of the same length, for example length 4 if $N = 15$.

Security of the Plain-RSA PKE scheme

Let $\mathcal{K}_{\text{rsa}}$ be an RSA generator. The plain RSA PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined via:

Alg $\mathcal{K}$	**Alg** $\mathcal{E}_{(N,e)}(M)$	**Alg** $\mathcal{D}_{(N,d)}(C)$
$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$ Return $((N, e), (N, d))$	$C \leftarrow M^e \bmod N$ return C	$M \leftarrow C^d \bmod N$ return M

Getting d from (N, e) involves factoring N.

But $\mathcal{E}$ is deterministic so we can detect repeats and the scheme is not IND-CPA secure.

The SRSA scheme

The SRSA PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ associated to RSA generator $\mathcal{K}_{\text{rsa}}$ and (public) hash function H: $\{0,1\}^* \to \{0,1\}^k$ encrypts k-bit messages via:

Alg $\mathcal{K}$

$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$
$ek \leftarrow (N, e)$
$dk \leftarrow (N, d)$
return (ek, dk)

Alg $\mathcal{E}_{N,e}(M)$

$x \xleftarrow{\$} \mathbf{Z}_N^*$
$K \leftarrow H(x)$
$C_a \leftarrow x^e \bmod N$
$C_s \leftarrow K \oplus M$
return (C_a, C_s)

Alg $\mathcal{D}_{N,d}(C_a, C_s)$

$x \leftarrow C_a^d \bmod N$
$K \leftarrow H(x)$
$M \leftarrow C_s \oplus K$
return M

Security of SRSA

The SRSA PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ associated to RSA generator $\mathcal{K}_{\mathrm{rsa}}$ and (public) hash function H: $\{0,1\}^* \to \{0,1\}^k$ can be proven IND-CPA assuming

- $\mathcal{K}_{\mathrm{rsa}}$ is one-way
- H is a "random oracle," meaning a "perfect" hash function.

In practice, $H(K)$ could be the first k bits of the sequence

$$\mathrm{SHA256}(0^8 \| K) \| \mathrm{SHA256}(0^7 1 \| K) \| \cdots$$

PKE summary

Scheme	**IND-CPA**?
DHIES	Yes
Plain RSA	No
SRSA	Yes
RSA OAEP	Yes

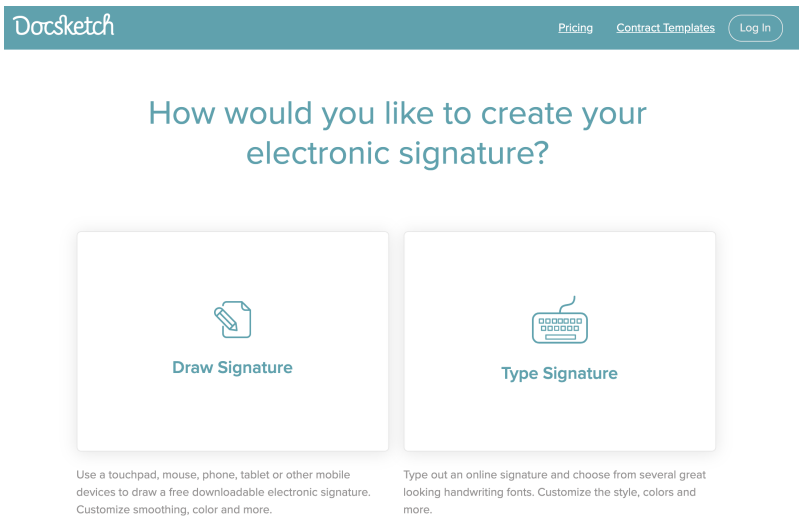
DIGITAL SIGNATURES

Signing by hand

Handwritten signatures are in common use to sign documents, checks, ...

Digitizing handwritten signatures

There are many ways to create digital versions of handwritten signatures.



Digitizing handwritten signatures

There are many ways to create digital versions of handwritten signatures.

Problem: A digitized handwritten signature is easily copied, leading to forgery.

To be secure, a digital signature must depend not only on the signer, but also on the message being signed.

Digital signatures syntax and correctness

A digital signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ consists of three algorithms that operate as follows:

- $(vk, sk) \xleftarrow{\$} \mathcal{K}$ — generate a verification key vk and matching signing key sk
- $\sigma \xleftarrow{\$} \mathcal{S}_{sk}(M)$ — apply signing algorithm $\mathcal{S}$ to signing key sk and message M to get a signature σ. Algorithm $\mathcal{S}$ may be randomized.
- $d \leftarrow \mathcal{V}_{vk}(M, \sigma)$ — apply verification algorithm $\mathcal{V}$ to verification key vk, message M and candidate signature σ to get a decision $d \in \{0, 1\}$.

Correctness requirement: $\Pr[\mathcal{V}_{vk}(M, \mathcal{S}_{sk}(M)) = 1] = 1$ for all (vk, sk) that may be output by $\mathcal{K}$ and all M in the underlying message space. The latter may depend on the verification key.

How it works

Step 1: Key generation
Alice locally computes $(vk, sk) \xleftarrow{\$} \mathcal{K}$ and stores signing key sk.

Step 2: Alice enables any prospective signature verifier to get vk.

Step 3: Alice can generate a signature σ of a message M using sk.

Step 4: Anyone holding vk can verify that σ is Alice's signature on M.

We don't require privacy of vk but we do require authenticity: the sender should be assured vk is really Alice's key and not someone else's. This can be done using the verification methods discussed in the previous lecture:

- Certificates and Public Key Infrastructure
- Out-of-band verification
- Bootstrap trusted infrastructure/bulletin boards/social media
- ...

Signatures versus MACs

Signatures accomplish the same security goals as MACs, but with public/private keys.

In a MAC, the signing and verifying keys are the same key K. Only entities that hold K can verify MACs, and verifiers can thus forge MACs.

With signatures, anyone holding vk can verify Alice's signature under sk. Verifiers cannot forge signatures.

Suppose Alice uses a MAC to sign checks, the key K shared between her and her bank. If the bank's servers are breached, and K is compromised, the adversary can forge Alice's checks.

If signatures are used instead, the bank holds Alice's verification key vk. The adversary recovering it cannot forge Alice's checks.

UF-CMA Security of a DS scheme

Intent: Adversary should not be able to get a verifier to accept σ as Alice's signature of M unless Alice has previously signed M, even if adversary can obtain Alice's signatures on messages of the adversary's choice.

A change from UF-CMA for MACs: Adversary gets the verification key.

As with MA schemes, the definition does not require security against replay. That is handled at the protocol level, via counters or time stamps.

UF-CMA game

Let $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a signature scheme and A an adversary.

Game $\text{UF-CMA}_{\mathcal{DS}}$

procedure Initialize
$(vk, sk) \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$
return vk

procedure Finalize(M, σ)
$d \leftarrow \mathcal{V}_{vk}(M, \sigma)$
return $(d = 1 \wedge M \notin S)$

procedure Sign(M)
$\sigma \xleftarrow{\$} \mathcal{S}_{sk}(M)$
$S \leftarrow S \cup \{M\}$
return σ

The uf-cma advantage of A is

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS}}(A) = \Pr\left[\text{UF-CMA}^{A}_{\mathcal{DS}} \Rightarrow \text{true}\right]$$

UF-CMA: Explanations

The "return vk" statement in **Initialize** means the adversary A gets the verification key vk as input. It does not get sk.

It can call **Sign** with any message M of its choice to get back a correct signature $\sigma \xleftarrow{\$} \mathcal{S}_{sk}(M)$ of M under sk. Notation indicates signing algorithm may be randomized.

To win, it must output a message M and a signature σ that are

- Correct: $\mathcal{V}_{vk}(M, \sigma) = 1$
- New: $M \notin S$, meaning M was not a query to **Sign**

Interpretation: Sign represents the signer and **Finalize** represents the verifier. Security means that the adversary can't get the verifier to accept a message that is not authentic, meaning was not already signed by the sender.

Plain RSA signature scheme

Let $\mathcal{K}_{\mathrm{rsa}}$ be an RSA generator. The plain RSA signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is defined via:

Alg $\mathcal{K}$	**Alg** $\mathcal{S}_{(N,d)}(y)$	**Alg** $\mathcal{V}_{(N,e)}(y, x)$
$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\mathrm{rsa}}$ Return $((N, e), (N, d))$	$x \leftarrow y^d \bmod N$ Return x	If $(x^e \bmod N = y)$ then return 1 Else return 0

Above, $vk = (N, e)$ is the verification key and $sk = (N, d)$ is the signing key.

The message space is $\mathbf{Z}_N^*$, and $y \in \mathbf{Z}_N^*$ is the message. The signature is $x \in \mathbf{Z}_N^*$.

Correctness: If $x \leftarrow \mathcal{S}_{(N,d)}(y)$ then

$$x^e \bmod N = (y^d)^e \bmod N = y^{ed \bmod \varphi(N)} = y^1 = y \ .$$

Security of plain RSA signatures

We let $f(x) = x^e \bmod N$ and $f^{-1}(y) = y^d \bmod N$.

To forge the signature $x = f^{-1}(y)$ of a message y, the adversary, given N, e but not d, must compute $y^d \bmod N$.

But the RSA generator $\mathcal{K}_{\mathrm{rsa}}$ is assumed OW-secure, so this task should be hard and the scheme should be secure.

Correct?

Security of plain RSA signatures

We let $f(x) = x^e \bmod N$ and $f^{-1}(y) = y^d \bmod N$.

To forge the signature $x = f^{-1}(y)$ of a message y, the adversary, given N, e but not d, must compute $y^d \bmod N$.

But the RSA generator $\mathcal{K}_{\mathrm{rsa}}$ is assumed OW-secure, so this task should be hard and the scheme should be secure.

Correct?

Of course not...

Attacks on plain RSA

adversary $A((N, e))$

Return $(1, 1)$

$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS}}(A) = 1$ because $1^d \bmod N = 1$

adversary $A((N, e))$

Pick some distinct $y_1, y_2 \in \mathbb{Z}^*_N \setminus \{1\}$
$x_1 \leftarrow \mathbf{Sign}(y_1)$; $x_2 \leftarrow \mathbf{Sign}(y_2)$
Return $(y_1 y_2 \bmod N, x_1 x_2 \bmod N)$

$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS}}(A) = 1$ because

$$(y_1 y_2)^d \bmod N = y_1^d y_2^d \bmod N = x_1 x_2 \bmod N$$

Other issues

In plain RSA, the message is an element of $\mathbb{Z}_N^*$. We really want to be able to sign strings of arbitrary length.

Full-Domain-Hash (FDH) [BR96]

Let $\mathcal{K}_{\text{rsa}}$ be an RSA generator. Let $\mathbf{H} : \{0,1\}^* \to \mathbf{Z}_N^*$. The RSA FDH signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is defined via:

Alg $\mathcal{K}$	**Alg** $\mathcal{S}_{(N,d)}(M)$	**Alg** $\mathcal{V}_{(N,e)}(M, x)$
$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$ Return $((N, e), (N, d))$	$y \leftarrow \mathbf{H}(M)$ $x \leftarrow y^d \bmod N$ Return x	$y \leftarrow \mathbf{H}(M)$ If $(x^e \bmod N = y)$ then return 1 Else return 0

Above, $vk = (N, e)$ is the verification key and $sk = (N, d)$ is the signing key.

The message space is $\{0,1\}^*$, and $M \in \{0,1\}^*$ is the message. The signature is $x \in \mathbf{Z}_N^*$.

Correctness: If $x \leftarrow \mathcal{S}_{(N,d)}(M)$ and $y \leftarrow \mathbf{H}(M)$ then

$$x^e \bmod N = (y^d)^e \bmod N = y^{ed \bmod \varphi(N)} = y^1 = y \ .$$

H needs to be collision resistant

Suppose we have an adversary C that can find a collision for **H**, meaning it returns (M_1, M_2) such that $M_1 \neq M_2$ but $\mathbf{H}(M_1) = \mathbf{H}(M_2)$. Then we can break the RSA FDH signature scheme $\mathcal{DS}$ via:

adversary $A((N, e))$

$(M_1, M_2) \xleftarrow{\$} C$; $x_1 \leftarrow \mathbf{Sign}(M_1)$; Return (M_2, x_1)

We have $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS}}(A) = 1$ because $\mathbf{H}(M_1) = \mathbf{H}(M_2)$ implies M_1, M_2 have the same signatures:

$$x_1 = \mathcal{S}_{(N,d)}(M_1) = \mathbf{H}(M_1)^d \bmod N = \mathbf{H}(M_2)^d \bmod N = \mathcal{S}_{(N,d)}(M_2)$$

Conclusion: UF-CMA security of RSA FDH requires that **H** be collision-resistant.

This condition is necessary for UF-CMA. But it is not by itself sufficient.

Instantiating **H** in RSA FDH

We seek suitable functions **H**: $\{0,1\}^* \to \mathbf{Z}_N^*$.

First let **G** be an XOF (eXtendable Output length Function), meaning it takes W, ℓ and returns ℓ bits. Possible choices are:

- $\mathbf{G}(W, \ell) = \mathrm{SHAKE256}(W, \ell)$
- $\mathbf{G}(W, \ell)$ is the first ℓ bits of the sequence

$$\mathrm{SHA256}(\langle 0 \rangle \| W) \,\|\, \mathrm{SHA256}(\langle 1 \rangle \| W) \,\|\, \cdots \,\|\, \mathrm{SHA256}(\langle 2^8 - 1 \rangle \| W)$$

where $\langle i \rangle$ is a 1-byte encoding of i and we assume $\ell \leq 2^8 \cdot 256$.

Instantiating **H** in RSA FDH

Let k be the security parameter associated to our RSA generator $\mathcal{K}_{\mathrm{rsa}}$, so that $2^{k-1} < N < 2^k$.

We could set $\mathbf{H}(M) = \mathbf{G}(M, k-1)$. However, this function is not onto $\mathbf{Z}_N$, since outputs y in the range $2^{k-1}, \ldots, N-1$ are never returned.

Instead we could set $\mathbf{H}(M) = \mathbf{G}(M, 2k) \bmod N$.

These functions return outputs in $\mathbf{Z}_N$, while we want outputs in $\mathbf{Z}_N^* \subseteq \mathbf{Z}_N$. We can simply ignore this, reasoning as follows.

First, for N a product of two distinct, odd primes, the RSA function $x \mapsto x^e \bmod N$ remains a permutation on $\mathbf{Z}_N$, with inverse $y \mapsto y^d \bmod N$, so correctness of the RSA FDH signature scheme is maintained.

Second, if an output $y = \mathbf{H}(M)$ is non-zero and in $\mathbf{Z}_N \setminus \mathbf{Z}_N^*$, then we can factor N, so this is unlikely to happen and security is maintained.

RSA PSS [BR96]

Let $\mathcal{K}_{\mathrm{rsa}}$ be an RSA generator with security parameter k, and ℓ a parameter satisfying $17 \leq 2\ell + 1 < k$. Let $\mathbf{H}_1, \mathbf{H}_2 : \{0,1\}^* \to \{0,1\}^{\ell}$ and $\mathbf{H}_3 : \{0,1\}^* \to \{0,1\}^{k-2\ell-1}$.

Example: $k = 2048$ and $\ell = 256$.

The key generation algorithm of the RSA PSS (Probabilistic Signature Scheme) $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is the usual one:

Alg $\mathcal{K}$

$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\mathrm{rsa}}$
Return $((N, e), (N, d))$

So $vk = (N, e)$ is the verification key and $sk = (N, d)$ is the signing key.

RSA PSS signing and verifying

Alg $\mathcal{S}_{(N,d)}(M)$

$r \xleftarrow{\$} \{0,1\}^{\ell}$
$w \leftarrow \mathbf{H}_1(M \| r)$
$r^* \leftarrow \mathbf{H}_2(w) \oplus r$
$y \leftarrow 0 \| w \| r^* \| \mathbf{H}_3(w)$
$x \leftarrow y^d \bmod N$
Return x

Alg $\mathcal{V}_{(N,e)}(M, x)$

$y \leftarrow x^e \bmod N$
$b \| w \| r^* \| P \leftarrow y$
$r \leftarrow r^* \oplus \mathbf{H}_2(w)$
If $(\mathbf{H}_3(w) \neq P)$ then return 0
If $(b = 1)$ then return 0
If $(\mathbf{H}_1(M \| r) \neq w)$ then return 0
Return 1

The message space is $\{0,1\}^*$, and $M \in \{0,1\}^*$ is the message. The signature is $x \in \mathbf{Z}_N^*$.

PSS standardization and usage

- RSA PKCS#1 v2.1, v2.2 / RFC 8017
- IEEE P1363a
- ANSI X9.31
- RFC 3447
- ISO/IEC 9796-2

Schnorr signature scheme

Let $G = \langle g \rangle$ be a cyclic group whose order m is a prime number. Let $\mathbf{H} : \{0,1\}^* \to \mathbf{Z}_m$. The Schnorr signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is defined via

Alg $\mathcal{K}$	**Alg** $\mathcal{S}^{\mathbf{H}}_x(M)$	**Alg** $\mathcal{V}^{\mathbf{H}}_X(M,(R,s))$
$x \xleftarrow{\$} \mathbf{Z}_m$	$r \xleftarrow{\$} \mathbf{Z}_m$; $R \leftarrow g^r$	If $(R \notin G)$ then return 0
$X \leftarrow g^x$	$c \leftarrow \mathbf{H}(R \Vert M)$	$c \leftarrow \mathbf{H}(R \Vert M)$
Return (X, x)	$s \leftarrow (r + cx) \bmod m$	If $(g^s = RX^c)$ then return 1
	Return (R, s)	Else return 0

Above, $vk = X$ is the verification key and $sk = x$ is the signing key.

The message space is $\{0,1\}^*$, and $M \in \{0,1\}^*$ is the message. The signature is $(R, s) \in G \times \mathbf{Z}_m$.

Correctness: If $(R, s) \xleftarrow{\$} \mathcal{S}_x(M)$ then

$$g^s = g^{r+cx} = g^r (g^x)^c = RX^c \ .$$

EdDSA signature scheme

EdDSA [BDLSY12] is a Schnorr-based signature scheme over an elliptic curve group.

Signing key sk is a random string of length a parameter b. It is expanded into a $2b$-bit string $x_1 \| x_2$. A clamping function is applied to x_1 to get the Schnorr signing key $x \in \mathbf{Z}_m$.

Signing is made deterministic by setting r to a hash of x_2 and the message.

There are several variants of the scheme.

These schemes are widely standardized, including RFC 8032 and FIPS 186-5. The scheme is used in many places including OpenSSH and GnuPG.

DSA

Another class of DL-based signatures evolved from the El Gamal signature scheme.

DSA (Digital Signature Algorithm) is a version over a group $\mathbf{Z}_p^*$ where p is a prime. It was proposed by NIST in 1991 and is in the FIPS 186-4 standard. However a draft of FIPS 186-5 indicates approval may not continue.

ECDSA (Elliptic Curve Digital Signature Algorithm) is a DSA variant over an elliptic curve group. It is also in FIPS 186-4.

Randomization in signatures

We say a signature scheme is randomized if its signing algorithm is randomized.

Randomized signature schemes include PSS, Schnorr, EdDSA, DSA/ECDSA.

Re-using coins (random choices) across different signatures is not secure, but there are (other) ways to make these schemes deterministic without loss of security. Namely, determine the coins (randomness) of the signing algorithm as a hash of the signing key and message.

Unlike for encryption, in signatures, randomness is not necessary for security.

The signature universe

Aggregate signatures, anonymous signatures, blind signatures, chameleon signatures, convertible undeniable signatures, delegatable signatures, forward-secure signatures, functional signatures, fuzzy signatures, group signatures, homomorphic signatures, identity-based signatures, invariant signatures, key-homomorphic signatures, leakage-resilient signatures, list signatures, malleable signatures, multi-signatures, online/offline signatures, partially-blind signatures, policy-based signatures, proactive signatures, redactable signatures, rerandomizable signatures, ring signatures, sanitizable signatures, structure-preserving signatures, threshold signatures, transitive signatures, undeniable signatures, unique signatures, ...