Security Architectures '18

✧

PKC Basics

Pierre Karpman
pierre.karpman@univ-grenoble-alpes.fr
https://www-ljk.imag.fr/membres/Pierre.Karpman/tea.html

2018–10–01

About this (small part of the) course

- (Re-) introducing a few public-key algorithms
- Illustrating the need for *Public-key Infrastructures* (PKIs)
- $\Rightarrow$ Some classical crypto stuff (some overlap w/ CRY-ENG?)

- Main course by F. Autréau & J.-G. Dumas

Public-key algorithms

Some major examples:

- Asymmetric encryption (one key to encrypt, another to decrypt), e.g. RSA (+ some randomized padding)
- Digital signature (one key to sign, another to verify), e.g. DSA
- Public-key key exchange, e.g. Diffie-Hellman

Note: RSA can be used to implement both a key-exchange and a signature

Diffie-Hellman etc.

RSA etc.

First things first: Diffie-Hellman

A simple protocol:

- Let $\mathbb{G} = \langle g \rangle$ be a cyclic finite group with a generator g
  - Example: $(\mathbb{Z}/512\mathbb{Z}, +)$, $g = 1$, $\mathrm{ord}(g) = 512$
  - Example: $\mathbb{F}_{257}^{\times}$, $g = 3$, $\mathrm{ord}(g) = 256$
  - Example: $(\mathbb{F}_2[X]/X^8 + X^4 + X^3 + X^2 + 1)^{\times}$, $g = X$, $\mathrm{ord}(g) = 255$
- A picks $a \xleftarrow{\$} \{0, \ldots, \mathrm{ord}(g) - 1\}$, sends g^a to B
- B picks $b \xleftarrow{\$} \{0, \ldots, \mathrm{ord}(g) - 1\}$, sends g^b to A
- A computes $(g^b)^a = g^{ba} = g^{ab}$, sets $k = \mathrm{KDF}(g^{ab})$
- B computes $(g^a)^b = g^{ab}$, sets $k = \mathrm{KDF}(g^{ab})$

With KDF some *key derivation function* (e.g. a ~ hash function)

Why this works?

Functionality

- A and B only need public information to perform the exchange
- They get the same k

$\Rightarrow$ Public-key key exchange

Security: necessary conditions

- Given g, g^a, g^b, it must be hard to compute g^{ab}
- $k = \mathrm{KDF}(g^{ab})$ must be “random-looking” when a, b are random
- There must be many possible values for k

Security focus

A necessary condition: computing *discrete logarithms* in $\mathbb{G}$ must be "hard"

Discrete logarithm

Let $\mathbb{G} = \langle g \rangle$ be a finite group of order N, the *discrete logarithm* of $h = g^a$, $a \in \{0, \ldots, N-1\}$ is equal to a

How hard is the "discrete logarithm problem" (DLP) for various groups?

DLP hardness

Proposition

It is always possible to compute the discrete logarithm in a group of order N in time $O(\sqrt{N})$

So one must *at least* pick N s.t. $2^{\log(N)/2}$ is large. But:

- $(\mathbb{Z}/n\mathbb{Z}, +)$: DLP always easy (logarithm ≡ division)
- $\mathbb{F}_q^\times$: usually hard, not *maximally* hard (needs much less than $\sqrt{N}$)
- $E(\mathbb{F}_q)$: usually maximally hard (needs about $\sqrt{N}$)

More on how to pick a group

If the order N of $\mathbb{G}$ is not prime, $\mathbb{G}$ has *subgroups*

- Let $N = pN'$, then g^p generates a group of order N'

Proposition (Pohlig-Hellman)

It is possible to solve the DLP in $\mathbb{G}$ subgroup-by-subgroup

$\Rightarrow$ For the DLP to be hard, $\mathbb{G}$ must be of order N s.t. DLP is hard in a subgroup of order p, the largest prime factor of N (But no details)

Are we done? Not quite

- Hardness of the DLP cannot be "proven", but a reasonable assumption for some groups
- We also need g^x to be random-looking (ditto)

But regardless, Diffie-Hellman as presented only protects againts *passive* adversaries

$\Rightarrow$ Not very useful in practice

Diffie-Hellman with a man in the middle

- A sends g^a to B
  - C intercepts the message, sends g^c to B
- B sends g^b to A
  - C intercepts the message, sends g^c to A
- A and C share a key $k_a = \mathrm{KDF}(g^{ac})$
- B and C share a key $k_b = \mathrm{KDF}(g^{bc})$
- Anytime A sends a message to B with key k_a, C decrypts and re-encrypts with k_b (and vice-versa)

One way to solve this: signatures

A wants to be sure it is talking to B

- Find B's public verification key for a signature algorithm
- Ask B to sign g^b
- Only accept it if the signature is valid

Works well, but A needs to know B's public key *beforehand*

$\Rightarrow$ We again have a bootstrapping issue

So are we back to square one?

Public-key infrastructures can help

Public keys still help compared to private ones:

- Possibly long term (v. have to be changed after a while (although not a real limitation))
- Scales linearly w/ the number of participants (v. quadratically)
- Trusting only one key is enough, if it signs all the ones you need

Example: TLS certificates

The simple picture:

- Web browsers are pre-loaded with “certificates” ($\sim$ public keys) of certification authorities (CAs)
- CAs sign the certificates of websites using secure connections (possibly using intermediaries)
- When connecting to a website, check the entire chain of certificates
- If everything’s fine, use the website’s public key to authenticate the exchange

More generally, we need a PKI!

So how do we sign?

Signature possibilities

- Use a discrete logarithm based protocol
- Or RSA
- But in both cases, also need a hash function!

Signatures: what?

Objectives of a signature algorithm:

- Given $(\mathsf{SK}, \mathsf{PK})$ a key pair
- message m + secret key SK$\leadsto$ signature $s = \mathsf{Sig}_{\mathsf{SK}}(m)$
- message m + signature s + public key PK$\leadsto$ verified message $\mathsf{Ver}_{\mathsf{PK}}(m, s)$

Informal security objectives

- Given PK, it should be hard to find SK
- Given PK, it should be hard to forge signatures
- (Variant: given access to a signing oracle $\mathbb{O}_{(\mathsf{SK},\mathsf{PK})}$, it should be hard to forge signatures)
- Formalised as *Existential unforgeability under chosen-message attacks* (EUF-CMA)

EUF-CMA for Public-Key signatures

EUF-CMA for $(\mathrm{Sig}, \mathrm{Ver})$: An adversary cannot forge a valid signature σ for a message m such that $\mathrm{Ver}(pk_C, \sigma, m)$ succeeds, when given (restricted) oracle access to $\mathrm{Sig}(sk_C, \cdot)$:

1. The Challenger chooses a pair (pk_C, sk_C) and sends pk_C to the Adversary
2. The Adversary may repeatedly submit queries m_i to the Challenger
3. The Challenger answers a query with $\sigma_i = \mathrm{Sig}(sk_C, m_i)$
4. The Adversary tries to forge a signature σ_f for a message $m_f \neq_i m_i$, s.t. $\mathrm{Ver}(pk_C, \sigma_f, m_f) = \top$

Related: interactive proof of identity

Objective of a proof of ID scheme:

- Publish public identification data α
- When challenged, prove knowledge of a secret related to α

Example of a one-time scheme:

1. Let $\mathcal{H}$ be a preimage-resistant hash function, $\mathcal{R}$ a large set
2. The prover draws $x \xleftarrow{\$} \mathcal{R}$, computes and publishes $X = \mathcal{H}(x)$
3. When challenged, reveals x

Many-time variant:

1. Draw $x \xleftarrow{\$} \mathcal{R}$, compute and publish $X = \mathcal{H}^N(x)$
2. When challenged, reveal $\mathcal{H}^{N-1}(x)$, reset $X = \mathcal{H}^{N-1}(x)$

A discrete-log based PoID scheme

~Schnorr identification scheme

1. Let $\mathbb{G} = \langle g \rangle$ be a group with a hard DLP
2. The prover draws $x \xleftarrow{\$} \mathcal{R}$, computes and publishes $X = g^x$
3. When challenged; draws r, sends $R = g^r$
4. The verifier picks c and sends it
5. The prover computes $a = r + cx$ and sends it
6. The verifier checks that $RX^c = g^a$

This can be run many times, BUT r's should be random and never repeat!

From PoID to signature

Differences between PoID and signatures:

- PoIDs are interactive (in the verification), signatures are not
- Signatures also involve a message

One major observation:

- If the prover can convince that it doesn't control *both* R and c, interaction is unnecessary
- (Otherwise, nothing is proved)

$\Rightarrow$ Fiat-Shamir transformation: generate c from R with a hash function

Schnorr signatures

To sign a message m with the key (SK, PK) pair $(x, X = g^x)$

1. Pick $r \xleftarrow{\$} \mathcal{R}$ and compute $R = g^r$
2. Compute $c = \mathcal{H}(R, m)$
3. Compute $a = r + cx$ and output (c, a) as the signature of m

To verify a signature:

1. Compute $\hat{R} = g^a / X^c = g^a / g^{cx}$
2. Check that $c = \mathcal{H}(\hat{R}, m)$

Important: r must (again) be random and not repeat! (Why?)

Where are we with dlog?

If $\mathbb{G} = \langle g \rangle$ is a prime-order group where the DLP is hard (on average $\equiv$ in the worst case), then:

- Can do asymmetric key exchange
- Can do public-key signatures

For signatures we also need

- Good hash functions
- Good pseudorandom number generation

Some comments on dlog attacks

When $\mathbb{G} \approx \mathbb{F}_p^\times$, the current dlog records are:

- $|p| \approx 768$ bits (Kleinjung et al., 2017), using a *Number Field Sieve* (NFS) algorithm
  - Took about 5300 core years
- $|p| \approx 1024$ bits for a *trapdoored* prime (Fried et al., 2017), using a *Special NFS* (SNFS) algorithm
  - Took about 385 core years

Note: it may be hard to decide if a prime is trapdoored

One nice (for an attacker) feature of (S)NFS:

- The largest part of the cost is a *precomputation*, then computing *individual dlogs* is *very fast*

Some more comments on dlog: small subgroup attack

Consider a *semi-static* key exchange,

- Where one of g^a or g^b (say g^b) is fixed

using $\langle g \rangle \subset \mathbb{F}_p^\times$ where $\mathbb{F}_p^\times$ has many small subgroups

- Then B must check that "$\hat{g}$" sent by A is in the correct group
- Otherwise, if $\hat{g}^b$ is in a small group of order N, a malicious A can learn $b \mod N$
- ... Then $b \mod N'$, etc.

One way to easily prevent this: use $p = 2q + 1$, q a Sophie Germain prime

$\Rightarrow$ Only a small subgroup of order 2 to check for in $\mathbb{F}_p^\times$

Diffie-Hellman etc.

RSA etc.

Back to basics

Greatest common divisor (GCD)

The *greatest common divisor* of two numbers a, $b \in \mathbb{N}$ is the largest number k, noted $\gcd(a, b)$ s.t. $a = km$, $b = km'$ for some m, $m' \in \mathbb{N}$

Co-primality

Two integers a, b are called *coprime* if $\gcd(a, b) = 1$

Examples:

- $\gcd(n, n) = \gcd(n, 0) = n$ for any n
- $\gcd(n, 1) = 1$ for any n
- $\gcd(n, kn) = n$ for any n
- $\gcd(p, q) = 1$ for any two prime numbers p, q
- $\gcd(p, n) = 1$ for any $n < p$

GCD computation

Given two integers, it is:

- Very important to be able to compute their gcd
- Very easy to do so (cool!)

$\rightsquigarrow$

A nice recurrence:

- Let $a,\ b \in \mathbb{N},\ a > b$
- Then $k = \gcd(a, b) = \gcd(b, a \mod b)$
  - If $a \mod b = 0$, then $a = kb = qb \Rightarrow \gcd(a, b) = \gcd(b, 0) = b$
  - If $a \mod b = r$, then $a = km = qb + r,\ b = km'$
  - $\Rightarrow km = qkm' + r \Rightarrow k(m - qm') = r \Rightarrow$ k divides r too!

Euclid's algorithm

The previous recurrence leads to Euclid's algorithm for gcd computation

GCD computation (recursive)

Input: a, $b < a$
Output: $\gcd(a, b)$

1. If $b = 0$, return a
2. Return $\gcd(b, a \mod b)$

In practice, iterative (variant) versions may be preferable

Extended Euclid

Let a, b, $k = \gcd(a, b)$

- Then for any u, $v \in \mathbb{Z}$, $ua + vb = ukm + vkm' = k(um + vm') = kw$ with $w = um + vm'$
- Of particular interest are any u, v s.t. $um + vm' = 1$, then we have $ua + vb = k = \gcd(a, b)$
- One can easily compute such u, v by *extending* Euclid's algorithm

Extended Euclid (cont.)

Extended Euclid algorithm

Input: a, $b < a$

Output: $k = \gcd(a, b)$, u, v s.t. $ua + vb = k$

1. If $b = 0$, return $(k = a, u = 1, v = 0)$ $\triangleright$ $1 \times a + 0 \times 0 = a$
2. Set $r = a \mod b$, $q = a \div b$ $\triangleright$ $r = a - qb$
3. Let $(k, u', v') \leftarrow \gcd(b, r)$ $\triangleright$ $u'b + v'r = k = \gcd(a, b)$
   $\triangleright$ $u'b + v'(a - qb) = k$
   $\triangleright$ $b(u' - q) + v'a = k$
4. Return $(k, v', u' - q)$

Applications: Dividing in $\mathbb{Z}/N\mathbb{Z}$

Let $a,\ b \in \mathbb{Z}/N\mathbb{Z}$, one wants to compute a/b

- Assuming we know how to multiply, we just need to compute b^{-1}
- To do this, compute u, v s.t. $ub + vN = 1 = \gcd(b, N)$
  - If $\gcd(b, N) > 1$, b is not invertible $\mod N$ (why?)
- Then $ub = 1 - vN \Rightarrow ub = 1 \mod N \Rightarrow u = b^{-1}$

Exercise: use this algorithm to prove that $\mathbb{Z}/N\mathbb{Z}$ is a field iff N is prime

Back to Crypto: RSA

RSA (Rivest, Shamir, Adleman, 1977) in a nutshell: a family of "one-way permutations with trapdoor"

- Publicly define $\mathcal{P}$ that everyone can compute
- Knowing $\mathcal{P}$, it is "hard" to compute $\mathcal{P}^{-1}$ (even on a single point)
- There is a *trapdoor* associated w/ $\mathcal{P}$
- Knowing the trapdoor, it is easy to compute $\mathcal{P}^{-1}$ everywhere

RSA: how?

- Let p, q be two (large) prime numbers
- Let $N = pq$
- Any $0 < x < N$ s.t. $\gcd(x, N) = 1$ is invertible in $\mathbb{Z}/N\mathbb{Z}$
  - Note that knowing $x \notin (\mathbb{Z}/N\mathbb{Z})^\times \Leftrightarrow$ knowing p and q
  - Why?

Proposition: order of $(\mathbb{Z}/N\mathbb{Z})^\times$

Let N be as above, the order of the multiplicative group $(\mathbb{Z}/N\mathbb{Z})^\times$ is equal to $(p-1)(q-1)$. (More generally, it is equal to $\varphi(N)$)

- So for any $x \in (\mathbb{Z}/N\mathbb{Z})^\times$, $x^{k\varphi(N)+1} = x$

RSA: more on how

- Let e be s.t. $\gcd(e, \varphi(N)) = 1$; consider $\mathcal{P} : x \mapsto x^e \mod N$
- $\mathcal{P}$ is a permutation over $(\mathbb{Z}/N\mathbb{Z})^\times$
- Knowing e, N, it is easy to compute $\mathcal{P}$
- Knowing e, $\varphi(N)$, it is easy to compute d s.t. $ed = 1 \mod \varphi(N)$
- Knowing d, x^e, it is easy to compute $x = x^{ed}$

$\Rightarrow$ We have a permutation with trapdoor, but how good is the latter?

RSA: how secure?

Knowing $ed = k\varphi(N) + 1$, it is easy to find $\varphi(N)$ (admitted)

Knowing $N = pq$, $\varphi(N) = (p-1)(q-1)$, it is easy to find p and q

- $\varphi(N) = pq - (p+q) + 1$; $p + q = -(\varphi(N) - N - 1)$
- For any a, b, knowing ab and $a + b$ allows to find a and b
  - Consider the polynomial $(X-a)(X-b) = X^2 - (a+b)X + ab$
  - $\Delta = (a+b)^2 - 4ab = (a-b)^2$
  - $a = ((a+b) + (a-b))/2$

$\Rightarrow$ Knowing, N, e, d, it is easy to factor N, plus:

- e does (basically) not depend on N

$\Rightarrow$ If it is easy to compute d from N, e, it is easy to factor N, and

- It is a hard problem to factor $N = pq$ when p, q are large random primes

BUT it might not be necessary to know d to (efficiently) invert $\mathcal{P}$

RSA for PKC

The objective: use RSA to build

- Public-key (asymmetric) encryption
  - Can then be used for asymmetric key exchange
- Public-key signatures

These schemes will need to satisfy the usual security notions

- For encryption: IND-CPA/CCA ("semantic security")
- For signatures: EUF-CMA

IND-CCA for Public-Key encryption

IND-CCA for $(\mathrm{Enc}, \mathrm{Dec})$: An adversary cannot distinguish $\mathrm{Enc}(pk_C, 0)$ from $\mathrm{Enc}(pk_C, 1)$, when given (restricted) oracle access to $\mathrm{Dec}(sk_C, \cdot)$ oracle:

1. The Challenger chooses a key pair (pk_C, sk_C), a random bit b, sends $c = \mathrm{Enc}(pk_C, b)$, pk_C to the Adversary
2. The Adversary may repeatedly submit queries $x_i \neq c$ to the Challenger
3. The Challenger answers a query with $\mathrm{Dec}(sk_C, x_i) \in \{0, 1, \perp\}$
   - This assumes w.l.o.g. that the domain of Enc is $\{0, 1\}$, and that decryption may fail
4. The Adversary tries to guess b

RSA Encryption: first attempt

Let $\mathcal{P}, \mathcal{P}^{-1}$ be RSA permutations with parameters N, e, d. Define:

- $\mathrm{Enc}(pk = (N, e), m) = \mathcal{P}(m) = (m^e \mod N)$
- $\mathrm{Dec}(sk = (N, e, d), c) = \mathcal{P}^{-1}(c) = (c^d \mod N)$

Not randomized $\Rightarrow$ fails miserably, not IND-CCA

- When receiving $c = \mathcal{P}(b)$, the Adversary compares with $c_0 = \mathcal{P}(0)$, $c_1 = \mathcal{P}(1)$

More issues with raw RSA

- If m, e are small, it may be that $m^e \mod N = m^e$ (over the integers) $\Rightarrow$ trivial to invert
  - Example: N is of 2048 bits, $e = 3$, m is a one-bit challenge: adding 512 random bits of padding before encrypting does not provide IND-CCA security!
- Consider a *broadcast* setting where m is encrypted as $c_i = m^3 \mod N_i$, $i \in [1,3]$. Suppose that $\forall i$, $m < N_i < c_i$. Using the CRT, one can reconstruct $m^3 \mod N_1 N_2 N_3 = m^3$ and retrieve m.
  - Even random padding might not prevent this attack, if too structured (Hastad, Coppersmith)

More issues with (semi-)raw RSA

A very useful result for analysing the security of RSA is due to Coppersmith (1996):

Finding small modular roots of univariate polynomials

Let P be a polynomial of degree k defined modulo N, then there is an efficient algorithm that computes its roots that are less than $N^{1/k}$

- The complexity of the algorithm is polynomial in k (but w. a high degree)
- Example application: if $c = (2^k B + a)^3 \mod N$ is an RSA image, B is known and of size $2/3 \log(N)$, one can find a of size $k < 1/3 \log(N)$ by solving $(2^k B + X)^3 - c = 0$ for X
- Other applications: in the previous slide; in slide #13, ...

Proper RSA-ENC

Let $\mathcal{P}, \mathcal{P}^{-1}$ be RSA permutations with parameters N, e, d. Let Pad, Pad^{-1} be a padding function and its inverse. Define:

- $\text{Enc}(pk = (N, e), m) = \mathcal{P}(\text{Pad}(m)) = (\text{Pad}(m)^e \mod N)$
- $\text{Dec}(sk = (N, e, d), c) = \text{Pad}^{-1}(\mathcal{P}^{-1}(c)) = \text{Pad}^{-1}(c^d \mod N)$

Necessary conditions on Pad:

- It must be invertible
- It must be randomized (with a large-enough number of bits)
- For all m, N, e, $\text{Pad}(m)^e$ must be larger than N

OAEP: A good padding function for RSA-ENC

OAEP: Optimal Asymmetric Encryption Padding (Bellare & Rogaway, 1994):

- Let $k = \lfloor \log(N) \rfloor$, κ be a security parameter
- Let $\mathcal{G} : \{0,1\}^\kappa \to \{0,1\}^n$, $\mathcal{H} : \{0,1\}^n \to \{0,1\}^\kappa$ be two hash functions
- Define $\mathsf{Pad}(x)$ as $(y_L \| y_R) = x \oplus \mathcal{G}(r) \| r \oplus \mathcal{H}(x \oplus \mathcal{G}(r))$, where $r \xleftarrow{\$} \{0,1\}^\kappa$
- One has $x = \mathsf{Pad}^{-1}(y_L \| y_R) = y_L \oplus \mathcal{G}(y_R \oplus \mathcal{H}(y_L))$

More on OAEP

- OAEP essentially uses a two-round Feistel structure
- To be instantiated, it requires two hash functions $\mathcal{H}$ and $\mathcal{G}$ with variable output size
- A possibility is to use a single XOF $\mathcal{X} : \{0,1\}^* \to \{0,1\}^*$, such as SHAKE-128

OAEP: Why does it work (kind of)?

Intuitively, full knowledge of $(y_L||y_R)$ is necessary to invert:

- If part of y_L is unknown, $\mathcal{H}(y_L)$, then $\mathcal{G}(y_R \oplus \mathcal{H}(y_L)$ are uniformly random
- If part of y_R is unknown, $\mathcal{G}(y_R \oplus \mathcal{H}(y_L))$ is uniformly random
- In both cases $\Rightarrow$ x is hidden by a “one-time-pad”

More formally, we would like a reduction of the form:

$$\text{Breaking RSA-OAEP w. Adv. } \varepsilon \Rightarrow \text{Inverting RSA w. Adv. } \approx \varepsilon$$

OAEP woes

- The original proof that OWP-OAEP is IND-CCA (for any good OWP) (Bellare & Rogaway, 1994) was incorrect
- Shoup showed that there can be no such proof (2001)
- But when OWP is RSA, then there *is* a proof (Shoup, 2001; Fujisaki & al., 2000)!
  - Exploits Coppersmith's algorithm!
- Not all the proofs are *tight* (e.g. Adv. $\varepsilon \Rightarrow$ Adv. ε^2)
  - Need large parameters to give a meaningful guarantee

What about RSA-SIG now?

Let $\mathcal{P}, \mathcal{P}^{-1}$ be RSA permutations with parameters N, e, d. Define:

- $\mathsf{Sig}(sk = (N, e, d), m) = \mathcal{P}^{-1}(m)$
- $\mathsf{Ver}(pk = (N, e), \sigma, m) = \mathcal{P}(\sigma) \texttt{ == } m \texttt{ ? } \top \texttt{ : } \bot$

Why this might work:

- Correctness: $(m^d)^e \equiv m \mod N$ $(\mathcal{P}^{-1} \circ \mathcal{P} = \mathcal{P} \circ \mathcal{P}^{-1} = \mathrm{Id})$
- Security: Comes from the hardness of inverting $\mathcal{P}$ w/o knowing d $\rightsquigarrow$ forging a signature for $m \Leftarrow$ compute $\mathcal{P}^{-1}(m)$

Raw RSA-SIG: That's no good!

- If $m \equiv m' \mod N$, then $\mathcal{P}^{-1}(m) = \mathcal{P}^{-1}(m') \Rightarrow$ trivial forgeries
- $\mathcal{P}^{-1}(m)\,\mathcal{P}^{-1}(m') = (m^d)(m'^d) \mod N = (mm')^d$ $\mod N = \mathcal{P}^{-1}(mm') \Rightarrow$ trivial forgeries over $[0, N-1]$

Again, some padding is necessary!

Proper RSA-SIG

Let $\mathcal{P}, \mathcal{P}^{-1}$ be RSA permutations with parameters N, e, d. Let Pad be a padding function. Define:

- $\mathrm{Sig}(sk = (N, e, d), m) = \mathcal{P}^{-1}(\mathrm{Pad}(m))$
- $\mathrm{Ver}(pk = (N, e), \sigma, m) = \mathcal{P}(\sigma)$ `==` $\mathrm{Pad}(m)$ `?` $\top$ `:` $\bot$

- Pad does not need to be invertible
- It does not need to be randomized (tho this can help)

What padding functions for RSA-SIG?

Let $k = \lfloor \log(N) \rfloor$

Full-Domain Hash (FDH) (Bellare & Rogaway; 1993):

- Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a hash function, $\mathrm{Pad}(m) = \mathcal{H}(m)$

PFDH (Coron, 2002):

- Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a hash function, $r \xleftarrow{\$} \{0,1\}^n$, $\mathrm{Pad}(m) = \mathcal{H}(m \| r)$
  - r is not included in the padding *per se*, but must be transmitted along

- Both are pretty simple, both provable in the random oracle model (ROM)
- The proof is *tighter* for PFDH ("good" security is obtained for smaller N)
- $\mathcal{H}$ can instantiated by a XOF

Another nice padding: PSS-R

PSS-R (Bellare & Rogaway, 1996):

- Let $\lfloor \log(N) \rfloor = k = k_0 + k_1 + k_2$, $\mathcal{H} : \{0,1\}^{k-k_1} \to \{0,1\}^{k_1}$, $\mathcal{G} : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1}$ be two hash functions, $r \xleftarrow{\$} \{0,1\}^{k_0}$
- Pad : $\{0,1\}^{k_2} \to \{0,1\}^{k}$ is defined by
  $\mathrm{Pad}(x) = \mathcal{H}(x \| r) \| (x \| r \oplus \mathcal{G}(\mathcal{H}(x \| r)))$
- If $|x| < k_2$, PSS-R is invertible (then, the message m does not need to be transmitted with the signature)
- Otherwise, e.g. compute $\mathrm{Pad}(x')$ where $x' = \mathcal{I}(x)$, $\mathcal{I} : \{0,1\}^* \to \{0,1\}^{k_2}$ a hash function (then, k_2 must be "large enough")

More on PSS-R

- In fact, PSS-R may also be used as padding for RSA-ENC (Coron & al., 2002)!
  - Notice the relative similarity between PSS-R and OAEP
- Both SIG and ENC cases are provably secure in the ROM
  - In the specific case of RSA, same as OAEP

RSA, DH recap, comparison

Roughly, hardness of factoring, DLOG $\Rightarrow$ Asymmetric key exchange, public-key signatures

- Factoring $\rightsquigarrow$ RSA: One-way permutation w. trapdoor, can be used for both
- DLOG $\rightsquigarrow$ DH, Schnorr/DSA/...: No permutation, but same functionalities

There are some differences, tho

Some DLOG schemes properties

- For key exchange, can change the secret every time $\Rightarrow$ "forward secrecy"
- For signatures, good randomness is essential! (Otherwise it breaks)
- Picking a random exponent is easy
- Picking a good group is not completely staightforward
- Some active attacks are possible
- It is possible to "break entire groups" (e.g. $\mathbb{F}_p^\times$)

Some RSA properties

- Secrets are fixed $\Rightarrow$ a break can compromise a long history
- No randomness needed for signatures (e.g. basic FDH), randomness failures don't reveal the secret
- Generating parameters is somewhat hard
- But all of them are independent (in principle)