Applied Cryptography
Spring Semester 2023
Lectures 27 and 28

Kenny Paterson (@kennyog)

Applied Cryptography Group

https://appliedcrypto.ethz.ch/

Overview

- Introduction to signatures
- Security of signatures: unforgeability and strong unforgeability
- Signatures in the discrete logarithm setting: DSA.
- RSA-based signatures: naïve RSA, full-domain hash RSA, RSA-PKCS and RSA-PSS
- Applications of signatures
- Some signature variants

Introduction

Introduction to signatures

- Assume Alice wants to send a message *m* to Bob.
- The adversary controls the network, as usual.
- The adversary would like to compromise the **integrity** of message *m.*
  - He wants Bob to receive and accept an alternative message *m'.*
- Alice and Bob need a cryptographic mechanism that detects modifications.
- Existing solution to this problem?
  - MAC algorithm
- What if Bob and Alice do not share a symmetric key?

Introduction to signatures – signing

- (Digital) signature schemes are a public key analogue of MAC algorithms.
- Alice has a signing key sk and a verification key vk.
- Bob is assumed to have an authentic copy of vk.
- We will discuss methods to distribute and authenticate keys in a later lecture: PKI and digital certificates.
- Alice uses an algorithm *Sign* to compute signatures on messages m:

$$\sigma = \mathrm{Sign}(sk, m)$$

- Alice sends (m, σ) to Bob in place of m.

Introduction to signatures – verification

- Bob uses an algorithm Vfy along with the verification key *vk* to verify the signature.
- Vfy algorithm outputs 0 or 1.
- Bob accepts *m* as having come from Alice if $\mathrm{Vfy}(vk, m, \sigma) = 1$ and rejects *m* otherwise.
- We need it to be hard for the adversary to find messages *m* and values σ such that $\mathrm{Vfy}(vk, m, \sigma) = 1$.
- That is, it should be hard for the adversary to *forge* signatures that verify using Alice's verification key.
- NB our formulation of signature schemes needs *m* as an input to the verification algorithm.
    - Some schemes provide *message recovery*: they can recover *m* (or part of *m*) from the signature σ during verification.

Formal definition of signature scheme

Definition:

A signature scheme $\mathcal{SIG}$ consists of a triple of algorithms (KGen,Sign,Vfy).

KGen is a randomised algorithm that outputs key pairs (*sk*,*vk*).

Sign takes as input *sk* and a message $m \in \{0, 1\}^*$, and outputs a signature σ.

Vfy takes as input a triple (vk,m,σ) and outputs 0 or 1.

Correctness:

For all key pairs (*sk*,*vk*) output by KGen, for all $m \in \{0, 1\}^*$, if σ = Sign(sk,m) then Vfy(vk,m,σ) = 1.

NB both Sign and Vfy may be randomised algorithms.

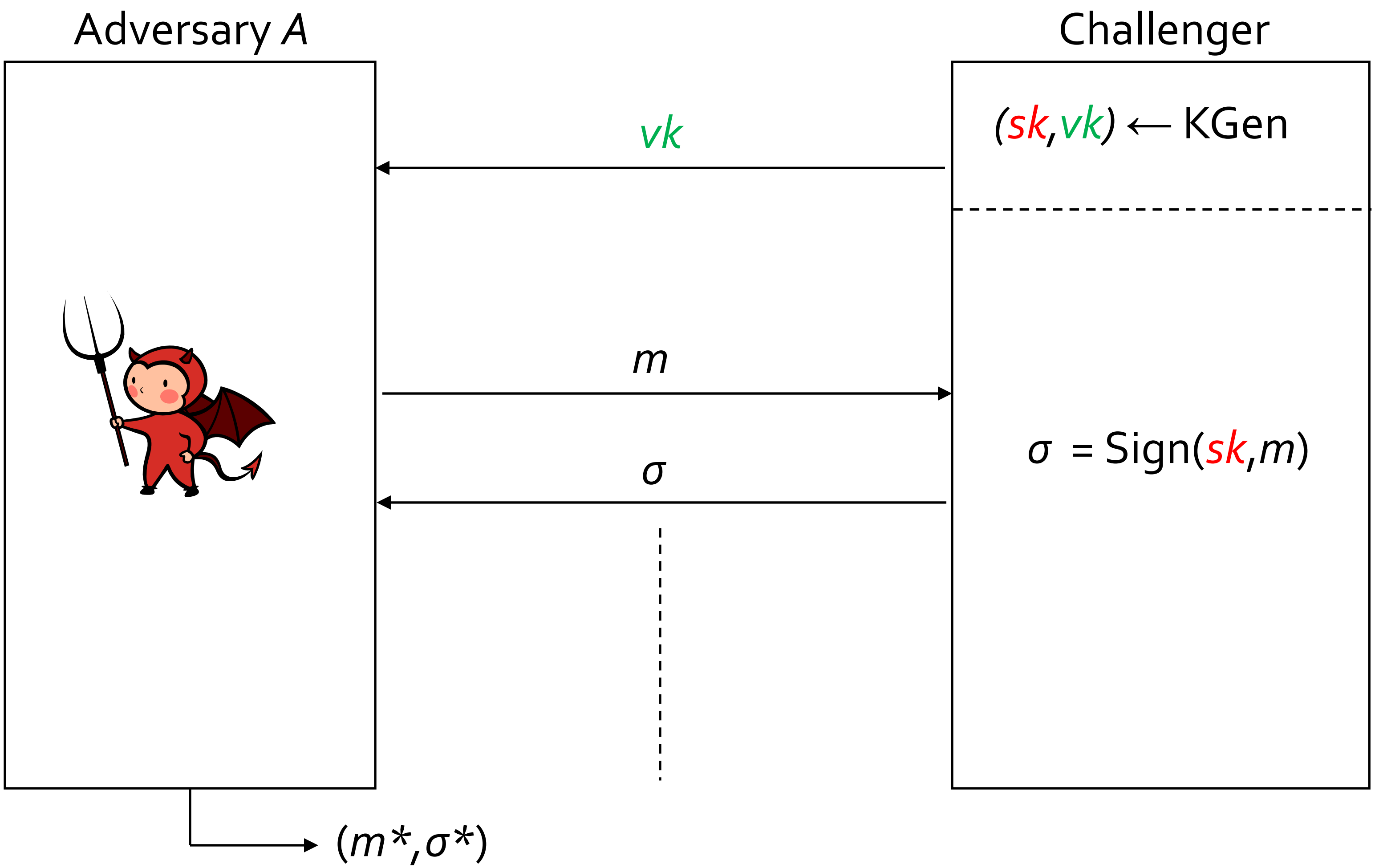
Security of signature schemes

Formal definition of security for signature schemes

- Security for a signature scheme $\mathcal{SIG}$ = (KGen,Sign,Vfy) is formalised in terms of a security game between a challenger and an adversary.
- Challenger generates a key pair (sk,vk) by running KGen.
- Challenger gives vk to adversary.
- Adversary runs, with access to a signing oracle: adversary sends m and gets back σ = Sign(sk,m).
- (No access to a verification oracle, cf. MAC security.)
- Adversary finally outputs (m^*,σ^*).
- Winning condition:

  Adversary wins if m^* is distinct from all the m queried to the signing oracle AND if Vfy(vk,m^*,σ^*) = 1.

UF-CMA for a signature scheme



$$\mathrm{Adv}_{SIG}^{UF\text{-}CMA}(A) := \Pr[\mathrm{Vfy}(vk,m^*,\sigma^*) = 1].$$

Formal definition of security for signature schemes

- Winning condition:

  Adversary A wins if m^* is distinct from all the m queried to the signing oracle AND if Vfy(vk,m^*,σ^*) = 1.

- The advantage of A is defined as:

$$\mathrm{Adv}_{\mathcal{SIG}}^{UF\text{-}CMA}(A) := \Pr[\mathrm{Vfy}(vk,m^*,\sigma^*) = 1].$$

- The scheme (KGen, Sign, Vfy) is said to be (q_S,t,ε)-UF-CMA secure if no adversary running in time t and making q_S queries to the signing oracle has advantage greater than ε.
- (E)UF = (existentially) unforgeable.
- CMA = chosen message attacks.

Consequences of the definition

UF-CMA security implies:

- It's hard for adversary find *sk* from *vk*. Why?
- It's hard for the adversary to create a forgery given just the verification key *vk*. Why?
- It's hard for the adversary to create a forgery when given access to signatures on random messages (that the adversary does not control). Why?
- If *vk* is properly bound to an identity, then the legitimate owner of the key pair (*vk*,*sk*) cannot deny having created a signature σ on a message *m*. Why?
  - This last property is usually called non-repudiation.
  - Can a MAC offer non-repudiation?

Strong unforgeability

- Strong unforgeability (SUF-CMA security):

  Adversary wins if (m^*,σ^*) is distinct from all the pairs (m,σ) involved in queries AND if $\mathrm{Vfy}(vk,m^*,\sigma^*) = 1$.

- Adversary could now win by producing a *new* signature σ^* on a message m that he previously queried to the signing oracle.
- This would not be a win in the UF-CMA game, but is a win in this game.
- It's now easier for the adversary to win, so the security notion is *at least as strong* as UF-CMA.
- Notions are equivalent for schemes with *unique* signatures.

Signatures in the discrete logarithm setting: DSA

Standards for signature schemes

NIST

- DSA, ECDSA, RSA-PSS, PKCS#1 v1.5 with RSA.

NSA Suite B

- ECDSA.

NESSIE

- ECDSA, RSA-PSS, ~~SFLASH~~ (broken).

CRYPTREC

- DSA, ECDSA, RSA-PSS, PKCS#1 v1.5 with RSA.

IEEE P1363

- DSA, ECDSA, PSS w/ RSA or Rabin-Williams, PKCS#1 v1.5 w/ RSA or Rabin-Williams.
- Some signature schemes with message recovery.

Reminder: The Discrete Log setting

- Let p, q be two prime numbers such that q divides p-1, and let g be an integer such that $g, g^2, g^3, \ldots g^q$ are distinct modulo p, and $g^q = 1$ mod p.
- The set $G_q = \{1, g, g^2, g^3, \ldots g^{q-1}\}$ of powers of g mod p is a cyclic group of prime order q with generator g.

The discrete logarithm problem in G_q:

Given p, q, g and an $y = g^x$ mod p from G_q, for a random value x in $\{0,1,\ldots,q-1\}$: find x.

- Diffie-Hellman key exchange and ElGamal encryption require a group where DLP is hard.

Digital Signature Algorithm (DSA)

- DSA was proposed by NIST in 1991.
- Explicitly required the use of a specific hash function
  - SHA-1.
- FIPS 186-4 updates to newer hash functions and larger key sizes.
- Variation of ElGamal signature scheme.
- Schnorr scheme is easier to prove secure but was patented .
- Very different set of functional capabilities compared to RSA:
  - DSA is a signature algorithm and cannot be easily converted into an encryption scheme.

DSA set up

- System parameters require two primes p and q:
  - e.g. 160-bit prime q, 1024-bit prime p so that $q \mid p-1$.
  - Other pairs of sizes: (224,2048), (256,2048), (256,3072).
  - Find g that generates G_q, as previously.

KGen:

1. Select random signing key x, $1 \leq x \leq q-1$.
2. Compute verification key $y = g^x \bmod p$.
3. Output (sk=x, vk=y).

- So the problem of finding signing key from verification key is an instance of the DLP.
- Many users can share the same system parameters (p,q,g).

DSA signing

To sign message m

1. hash message m to get $H(m)$
2. generate random value k, $1 \leq k \leq q\text{-}1$
3. compute $r = (g^k \bmod p) \bmod q$
4. compute $k^{-1} \bmod q$
5. compute $s = k^{-1}(H(m) + x \cdot r) \bmod q$
6. output $\sigma = (r,s)$

- Signatures can be represented using 2 x 160 = 320 bits (DSA signatures are much smaller than RSA signatures at same security level).
- Signing requires one exponentiation mod p using an exponent k of 160 bits (a short exponent)

DSA verification

To verify that $\sigma = (r,s)$ is a signature for message m:

1. check that $1 \leq r \leq q-1$ and $1 \leq s \leq q-1$
2. compute $w = s^{-1} \bmod q$
3. compute $u_1 = w \cdot H(m) \bmod q$ and $u_2 = w \cdot r \bmod q$
4. accept signature if the following equation holds

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

Correctness: suppose $\sigma = (r,s)$ is a signature for message m. Then:

$$g^{u_1} y^{u_2} = g^{s^{-1} H(m)} \cdot y^{r\, s^{-1}} = g^{s^{-1}(H(m)+xr)} = g^{k} \bmod p$$

and so

$$(g^{u_1} y^{u_2} \bmod p) \bmod q = (g^{k} \bmod p) \bmod q = r.$$

Security of DSA signature scheme

Informal:

- Attacks extracting the private key:
  - Solving DLP mod p.
  - $O(q^{1/2})$ attacks in the subgroup G_q of order q.
  - Need to choose p and q both large enough to prevent these attacks.
- Hash function collisions: $H(m_1) = H(m_2)$ implies: if $\sigma = (r,s)$ is a valid signature for message m_1 then it is also valid for m_2.

Formal:

- No clean security proof for DSA is known.
- There are various proofs under different assumptions and heuristics of varying strength; none is really satisfactory.

Security of DSA under randomness failure

- Suppose the same value k is used with a key x to sign two different messages, m_1 and m_2, giving signatures $\sigma_1 = (r_1, s_1)$ and $\sigma_2 = (r_2, s_2)$.
- Then $r_1 = (g^k \bmod p) \bmod q = r_2$.
- So the "repeated k" condition is detectable from signatures alone.
- Moreover:

$$s_1 = k^{-1}(H(m_1) + x \cdot r) \bmod q \qquad \text{and} \qquad s_2 = k^{-1}(H(m_2) + x \cdot r) \bmod q.$$

- So:

$$s_1 - s_2 = k^{-1}(H(m_1) - H(m_2)) \bmod q.$$

- Hence:

$$k = (s_1 - s_2)^{-1} \cdot (H(m_1) - H(m_2)) \bmod q.$$

- From k, we can recover x, the **private signing key**, by solving the equation:

$$s_1 = k^{-1}(H(m_1) + x \cdot r) \bmod q$$

  using the known values s_1, k, $H(m_1)$, r.

- So repeating k values leads to a catastrophic security failure.

Security of DSA under randomness failure

The above key recovery issue in DSA and the related ECDSA scheme has occurred regularly in practice!

- OpenSSL bug in Debian (2008):

  "It was discovered that the RBG in Debian's openssl package is predictable. […] It is strongly recommended that all cryptographic key material is recreated from scratch. […] All DSA keys ever used on affected systems for signing should be considered compromised."

- Hackers recovered Sony's PlayStation 3 signing key (2010).
- Bad random number generator in Android allowed Bitcoins to be stolen (2013).
- The problem occurs naturally in virtualized environments.

Hedging DSA against randomness failures

- Related, but more complicated attacks are possible if only some *bits* of the random values k can be predicted.
- This can happen in a timing attack setting, e.g. signing may be faster if MSBs of k are zero.
- Similarly, problems if one uses certain weak generators to produce k, or if some relations between the bits of k are known.
- We can hedge against randomness failures by *derandomising*:
- Generate k in signing using a pseudo-random function F with a key K:

$$k = F_K(vk \,||\, m).$$

- Ensures different random(-looking) values for different verification keys and messages.
- Need to keep PRF key K as part of signing key.
- See RFC 6979 for related scheme ECDSA; technique applies more generally for randomised signature schemes.

Psychic Signatures

- During verification, it is essential to check that $1 \leq r \leq q-1$ and $1 \leq s \leq q-1$
- Failure to do so can allow trivial forgery attacks on DSA and ECDSA.
- It was recently discovered that Java versions 15-18 failed to do these checks for ECDSA.
- Earlier versions of Java were not vulnerable.
- So what happened?
- https://neilmadden.blog/2022/04/19/psychic-signatures-in-java/
- https://www.oracle.com/security-alerts/cpuapr2022.html

RSA-based signatures

RSA-based signatures

First attempt: naïve RSA-based signatures:

DO NOT USE!

KGen: as in RSA encryption, we set $vk = (N,e)$ and $sk = d$, where $N = pq$ is a product of two large primes and $ed = 1 \bmod (p-1)(q-1)$

Sign: we use the "private key" $sk = d$ to sign:

$$\sigma = m^d \bmod N.$$

Vfy: given (m,σ), we check whether $\sigma^e = m \bmod N$.

Security?

Given signatures σ_1 and σ_2 on messages m_1 and m_2, simple algebra shows that $\sigma_1\sigma_2 \bmod N$ is a valid signature on $m_1 m_2$.

This makes creation of forgeries trivial.

Exercise: formulate the attack in the context of the UF-CMA security game.

RSA Full Domain Hash (FDH) signatures

Second attempt: full-domain hash RSA signatures:

KGen: as in RSA encryption, we set $vk = (N, e)$ and $sk = d$, where $N = pq$ is a product of two large primes and $ed = 1 \bmod (p-1)(q-1)$.

Sign: we use the "private key" $sk = d$ to sign:

$$\sigma = H(m)^d \bmod N,$$

where H is a hash function from $\{0,1\}^*$ into $\{0,\ldots,N-1\}$ ("full domain").

Vfy: given (m,σ), we check whether $\sigma^e = H(m) \bmod N$.

Security?

Use of a hash function destroys the multiplicative structure that enabled the previous attack.

Also allows signing of long messages.

Needs a collision-resistant hash function.

RSA FDH signatures

Theorem

RSA-FDH is UF-CMA secure under the assumption that RSA inversion is hard and *H* is a random oracle.

More precisely, for any adversary *A* against UF-CMA security of RSA-FDH, there exists an adversary *B* against RSA inversion such that:

$$\mathrm{Adv}_{\text{RSA-FDH}}^{\mathit{UF\text{-}CMA}}(A) \leq (q_s+q_h) \cdot \mathrm{Adv}_{\text{RSA-INV}}(B) - 1/N.$$

where q_s is the number of signing queries and q_h is the number of hash queries made by *A.* Moreover, *B* runs in (roughly) the same time as *A.*

NB A tighter result can be proved, with q_s+q_h replaced by q_s, see Coron (CRYPTO 2000).

Hash-based RSA signatures with padding

Third attempt: RSA signatures with padding:

Sign: we use the "private key" $sk = d$ to sign:

$$\sigma = \text{pad}(H(m))^{d} \mod N,$$

where H is a collision-resistant hash function with short output (e.g. SHA256) and pad(.) is some deterministic padding scheme.

Vfy: given (m,σ), we check whether $\sigma^{e} = \text{pad}(H(m)) \mod N$.

This approach is widely standardised and still in common use, e.g. PKCS#1 v.1.5 standard, ANSI X9.31, IEEE P1363a, SSL/TLS, IPsec, EMV.

PKCS#1 v.1.5 padding: 00 01 FF… FF 00 || c || $H(m)$, with constant c.

Hash-based RSA signatures with padding

Security?

- No security proof for the scheme with PKCS#1 v.1.5 padding is known.
- Signatures can be forged if the constant part of the padding is too short.
- Padding check/removal often wrongly implemented, see e.g. Bleichenbacher attack described in Boneh-Shoup, Section 13.6.1.
- A proof for a closely related, but distinct, scheme was given in:

  Tibor Jager, Saqib A. Kakvi, Alexander May: On the Security of the PKCS#1 v1.5 Signature Scheme. ACM CCS 2018: 1195-1208.

RSA-PSS

Fourth attempt: RSA Probabilistic Signature Scheme (RSA-PSS):

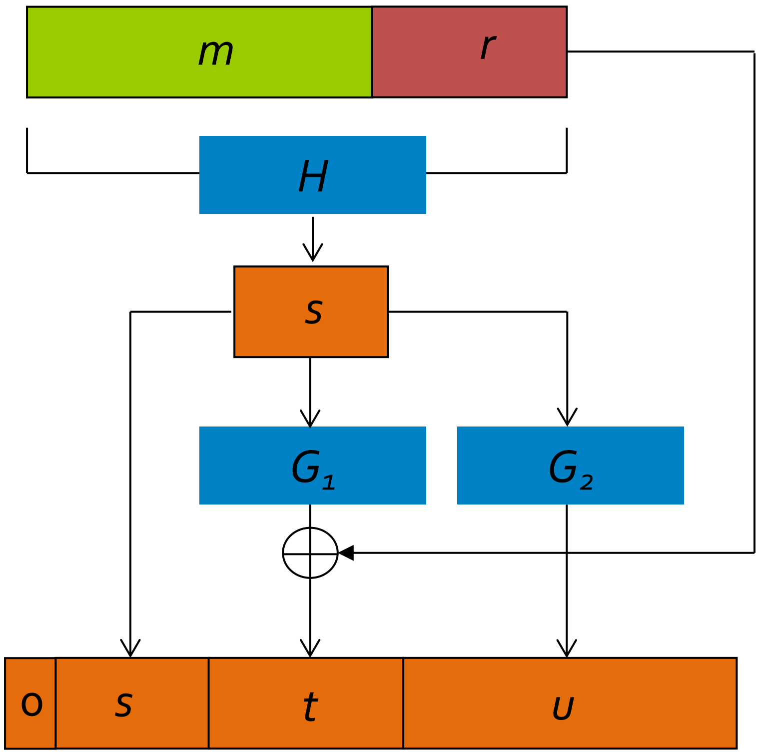
KGen: as in RSA encryption, we set $vk = (N, e)$ and $sk = d$, where $N = pq$ is a product of two large primes and $ed = 1 \bmod (p-1)(q-1)$.

Sign (simplified):

1. Generate a random value r (with 256 bits, say).
2. Compute $s = H(m||r)$, $t = G_1(s) \oplus r$, $u = G_2(s)$ where G_1, G_2 and H are suitable hash functions (the sum of their output lengths should be λ-1, where λ is bit-length of N).
3. Output:

$$\sigma = (0 \,||\, s \,||\, t \,||\, u)^d \bmod N,$$

RSA-PSS (simplified) encoding



RSA-PSS verification

Vfy (simplified): given (m,σ):

1. Compute $\sigma^e \bmod N$ and parse the result as $b \,||\, s \,||\, t \,||\, u$ where b is a bit and s, t, u have the correct lengths (determined by output lengths of H, G_1, G_2).
2. Output "0" if $b = 1$.
3. Compute r' by inverting the equations for s, t:
   - Compute $r' = G_1(s) \oplus t$
4. Re-encode r' and m and check correctness:
   - compute $s' = H(m||r')$, $t' = G_1(s') \oplus r'$, $u' = G_2(s')$.
   - check $(s' = s) \wedge (u' = u)$; output "0" if this fails; otherwise output "1".

RSA-PSS

Security of RSA-PSS:

Assuming G_1, G_2 and H behave like random functions, the UF-CMA security of RSA-PSS can be **tightly** related to the hardness of the RSA inversion problem.

- Proof in a paper by Bellare-Rogaway (CRYPTO 1996).
- RSA-PSS can be instantiated with "ordinary" collision-resistant hash functions, e.g. SHA-256 (no hashing onto full domain is needed).
- RSA-PSS is standardised in PKCS#1 v2.1, IEEE P1363a-2004, IETF RFC 3447,…
- **If you have to use RSA signatures, then RSA-PSS is the right choice of scheme.**

Summary of RSA-based signatures

Naïve RSA: DO NOT USE UNDER ANY CIRCUMSTANCES.

Hash-then-sign: no known attack, but also no proof.

This includes basic hash-then-sign, and PKCS#1 v.1.5 padding.

The latter was standardized early and still in widespread use.

Full-Domain Hash: provably secure, with weak proof.

RSA-PSS: provably secure, with tight proof. Please use!

Applications of signatures

Applications of signatures

- Public verification of message authenticity/integrity.
- Code-signing.
- Proof of ownership/transfer of cryptocurrency.
- Entity authentication and identification.
  - Sign challenge messages to prove possession of a signing key matching a verification key.
  - Used as a building block in more complex protocols such as SSL/TLS.
- Certification systems, public key infrastructures (PKI).
  - Use signatures to authenticate other signing keys and public encryption keys.
  - See Boneh-Shoup Chapter 13.8 for a good discussion.

Practical challenges

- It is often thought that cryptographic signatures can replace handwritten signatures.
  - The required legal frameworks in place worldwide, e.g., European directive 1999/93/EC.
  - But typically high demands on physical security (key storage, signature generation), requirements for tamperproof hardware (e.g. smartcards) and special terminals to interact with it.
  - Deployments as part of electronic identity cards in some countries, e.g. Belgium, Estonia.
- Human understanding and usability of software are major barriers to wide adoption.
- Management of keys via suitable infrastructure.
- Making robust implementations that avoid security pitfalls, e.g. bad randomness.

Some signature variants

Some signature variants

- **Blind signatures**: an interactive protocol in which *A* sends *B* a blinded message to sign, *B* learns nothing about the message, and *A* obtains a regular signature – used in anonymous credential systems.
- **Group signatures**: anyone from a group of users can sign; no-one can tell who signed, except possibly a "group manager" who can reveal the signer – used in TCG TPMs.
- **Threshold signatures**: any *k* out of *n* parties can sign a message that verifies under some key *vk*, but *k-1* or fewer cannot - used in cryptocurrency custody solutions.
- **Proxy signatures**: limited signing authority can be delegated to another party (a proxy).
- Also: ring signatures, multi-signatures, aggregate signatures,….

Homework

- Read Chapter 13 of Boneh-Shoup for many more details, constructions and proofs.
- Chapter 14 of Boneh-Shoup treats hash-based signatures in detail.
- Start next exercise sheet.
- Prepare for this week's lab.

Extra slides

RSA FDH signatures: Interpreting the security bound

The security bound is of the form:

$$\mathrm{Adv}^{\mathit{UF\text{-}CMA}}_{\text{RSA-FDH}}(A) \le (q_s+q_h) \cdot \mathrm{Adv}_{\text{RSA-INV}}(B) - 1/N.$$

Let's ignore the $1/N$ term (it's usually very small).

Suppose we choose RSA parameters such that the following is plausible:

$$\mathrm{Adv}_{\text{RSA-INV}}(B) = 2^{-128}$$

for any adversary B running in a reasonable amount of time (e.g. 3072-bit N).

Suppose $q_s+q_h = 2^{80}$, thinking of an adversary that can perform many hash computations (but can perhaps make far fewer signing queries).

Then the bound says:

$$\mathrm{Adv}^{\mathit{UF\text{-}CMA}}_{\text{RSA-FDH}}(A) \le 2^{80} \cdot \mathrm{Adv}_{\text{RSA-INV}}(B) \le 2^{-48}.$$

Hence the factor (q_s+q_h) becomes significant in assessing what security guarantees we actually get for the scheme, and how we should choose our scheme parameters in practice.

This is why **tight** security reductions are preferred if we can obtain them.

RSA FDH signatures: Interpreting the security bound

In Coron's improved analysis, the security bound is of the form:

$$\mathrm{Adv}_{\text{RSA-FDH}}^{\textit{UF-CMA}}(A) \leq q_s \cdot \mathrm{Adv}_{\text{RSA-INV}}(B)$$

Now suppose $q_s \leq 2^{32}$, thinking of an adversary that can only make a limited number of signing queries (since these are typically made online, require interaction, and can be rate-limited).

Assuming again that $\mathrm{Adv}_{\text{RSA-INV}}(B) = 2^{-128}$, now the bound says:

$$\mathrm{Adv}_{\text{RSA-FDH}}^{\textit{UF-CMA}}(A) \leq 2^{32} \cdot \mathrm{Adv}_{\text{RSA-INV}}(B) \leq 2^{-96}.$$

Clearly this is much better from a security perspective, though we still don't get the full 128-bit security despite using a 3072-bit modulus *N.*

NB A full analysis would also take into account running times in a more detailed way.

RSA FDH signatures: Security proof

Recall: RSA inversion problem: given (N,e,x) where x is uniformly random modulo N, compute x^d mod N (where $de = 1$ mod $(p-1)(q-1)$).

Sketch proof:

- B receives as input (N,e,x). B gives (N,e) to A as the public key.
- A makes two kinds of query: signing queries and hash queries.
- For every signing query made by A, adversary B will also make a hash query.
- So let the hash queries be on inputs m_i, $1 \leq i \leq q_s + q_h$.

RSA FDH signatures: Security proof

Sketch proof (ctd):

- B picks a random i^* in $\{1,.., q_s + q_h\}$ at the start of the game and sets $H(m_{i^*}) = x$ when query i^* is made.
- For all other i, B sets $H(m_i) = y_i^e$ mod N where $y_i \leftarrow_\$ \{0,\dots,N\text{-}1\}$.
- Note that y_i is then a valid signature on m_i for all $i \neq i^*$:

$$y_i = y_i^{ed} = H(m_i)^d \bmod N.$$

- Moreover, if A outputs a valid forgery (m_{i^*},σ^*), then:

$$(\sigma^*)^e = H(m_{i^*}) = x \bmod N$$

so that:

$$\sigma^* = (\sigma^*)^{ed} = H(m_{i^*})^d = x^d \bmod N$$

and hence σ^* is a solution to the RSA inversion problem for input (N,e,x).

RSA FDH signatures: Security proof

Sketch proof (concluded):

- B also needs to handle A's signing queries.
- This is now straightforward: if A makes a signing query on some input m_i, then B makes a hash query on m_i for itself to get y_i^e, and can now produce forgery y_i.
- Tricky case: if A makes a signing query on m_{i*}: then B simply aborts.
- B wins if A wins on output $(m_{i*},\sigma*)$, i.e. if A forges on the $i*$-th query to H; this event happens with probability $1/(q_s + q_h)$.
- Missing part of analysis: we need to show that to be successful at all, A must make a query to H on its chosen message $m*$ (but not make a signing query on this message).
- This follows from the fact that, unless A makes this query, H is still uniformly random on $m*$, so then A would be successful with probability only $1/N$.