Digital Signatures

CMA definitions, Fiat-Shamir transform, RSA-FDH, Schnorr, DSA, Lamport, Signcryption, Group sig., etc.

Signature Uses

- Used to provide integrity, authenticity in public-key setting
- E.g., company sending periodic updates to software on clients
  - (PK_C, SK_C); PK_C embedded in original software
  - (U, $\sigma = \mathrm{Sign}_{SK_C}(U)$) sent to client
  - Client runs Verify(PK_C, σ, U) $\stackrel{?}{=}$ "accept"

- Intuitively, don't want A to:
  - Produce U′, s.t., Verify(PK_C, σ, U′) = "accept", and/or
  - Produce σ', s.t., Verify(PK_C, σ', U) = "accept"

Logistical Issues

- Signature replay attacks possible (use timestamps, nonces)

- Need to have reliable transmission/distribution mechanism for PK
- PKI, CAs,…

Signatures vs. MACs

- Both provide integrity, but MAC in shared-key setting
- Pros of Signatures
  - Setup once, even for multiple receivers
  - Publicly verifiable
  - Transferable
  - Non-repudiable
- Cons:
  - MACs faster, $|\text{MACs}| < |\text{Signatures}|$
  - In single-recipient case, MACs better

Basic Definition

- A signature scheme consists of 3 algorithms:
  - (PK,SK) <— KeyGen(1^n): randomized
  - σ <— $\mathrm{Sign}_{SK}(m)$: randomized
  - {"accept","reject"} <— $\mathrm{Verify}_{PK}(m,\sigma)$: deterministic

- Message-spaces, and signature-spaces well-defined
- E.g., $m,\sigma \in G$, $|G| = q$, for prime q, etc.

- If σ <— $\mathrm{Sign}_{SK}(m)$, then "accept"<— $\mathrm{Verify}_{PK}(m,\sigma)$, except with negl. probability, for all legal messages m, and well-formed (PK,SK)

Standard CMA Model

- Standard EUF-CMA: similar to CCA2

- Signature Forgery Game for scheme Π, adversary A:
  - Challenger runs KeyGen(1^n) —> (PK,SK)
  - A given PK, *adaptively* requests q signatures[1]: $(m_1,\ldots,m_q) \in \{0,1\}^n$, gets $(\sigma_1,\ldots,\sigma_q)$, where $\sigma_i \leftarrow \mathrm{Sign}_{SK}(m_i)$
  - A outputs (m^*, σ^*)
  - A wins if ($m^* \notin (m_1,\ldots,m_q)$ AND $\mathrm{Verify}_{PK}(m^*,\sigma^*)$ = accept). Set game output = 1
- Π is *existentially unforgeable* against *adaptive chosen message* attacks (EUF-CMA) if for all PPT adversaries A, there is a negl. Function, s.t.,

$$\Pr[\mathrm{ForgeryGame}_{A,\Pi}(n) = 1] \leq \mathrm{negl}(n)$$

1: On messages of A's choice, that is

Weak CMA

- EUF-WeakCMA — adversary submits all messages before seeing PK

- Signature Forgery Game for scheme Π, adversary A:
  - A sends challenger $(m_1,\ldots,m_q) \in \{0,1\}^n$
  - Challenger runs KeyGen(1^n) —> (PK,SK), creates signatures $(m_1,\sigma_1),\ldots,(m_q,\sigma_q)$, where σ_i<— $\mathrm{Sign}_{SK}(m_i)$
  - A given PK, $(m_1,\sigma_1),\ldots,(m_q,\sigma_q)$
  - A outputs (m^*, σ^*)
  - A wins if ($m^* \notin (m_1,\ldots,m_q)$ AND $\mathrm{Verify}_{PK}(m^*,\sigma^*)$ = accept). Set game output = 1
- Π is *existentially unforgeable* against *weak chosen message* attacks (EUF-CMA) if for all PPT adversaries A, there is a negl. Function, s.t.,

$$\Pr[\mathrm{ForgeryGame}_{A,\Pi}(n) = 1] \leq \mathrm{negl}(n)$$

Strong/Full CMA

- Strong EUF-CMA — requires A cannot produce (valid) new signature even on a previously signed message

- Signature Forgery Game for scheme Π, adversary A:
  - Challenger runs KeyGen(1^n) —> (PK,SK)
  - Proceeding adaptively, A requests q signatures: $(m_1,\ldots,m_q) \in \{0,1\}^n$, gets $(\sigma_1,\ldots,\sigma_q)$, where σ_i <— $\text{Sign}_{SK}(m_i)$
  - A outputs (m^*, σ^*)
  - A wins if $(((m^*,\sigma^*) \notin (m_1,\sigma_1),\ldots,(m_q,\sigma_q))$ AND $(\text{Verify}_{PK}(m^*,\sigma^*) =$ accept)). Set game output = 1
- Π is strongly *existentially unforgeable* against *adaptive chosen message* attacks (EUF-CMA) if for all PPT adversaries A, there is a negl. Function, s.t.,

$$\Pr[\text{ForgeryGame}_{A,\Pi}(n) = 1] \leq \text{negl}(n)$$

Unforgeability

- *Existential Forgery*: A forges a signature for at least one message, has no control over the message
- *Selective Forgery*: A forges a signature for a particular message chosen by her
- *Universal Forgery*: A finds an efficient signing algorithm that can forge signatures on any message(s)
- *Total Break*: A finds signer's signing key

- Max. level of security is against *Existential Forgery*: If A can't do even this, she can't do anything harder either…
- Lowest level of security – security against *total break*: Ability to do a total break implies A can do SF, UF and EF too
- The most secure signature schemes are *existentially unforgeable (EUF)*

Hash-and-Sign

- A hashed signature scheme consists of 3 algorithms:

  - (PK,SK) <— $Gen(1^n)$
    - (pk,sk) <— $KeyGen(1^n)$
    - s <— $HashGen(1^n)$
    - Set PK = (pk,s), SK = (sk,s)
  - σ <— $Sign_{SK}(m \in \{0,1\}^*)$
    - Compute σ <— $Sign_{SK}(H^s(m))$
  - {“accept”,”reject”} <— $Verify_{PK}(m,\sigma)$
    - If $Verify_{PK}(H^s(m),\sigma) \stackrel{?}{=}$ “accept”, return 1

Hash-and-Sign

- Correctness property easily carries over

- Security too carries over
  - Informally, if sig. scheme is (standard or strong/weak) EUF-CMA, and H is collision resistant, hash-and-sign paradigm is secure, for $m \in \{0,1\}^*$ (arbitrary-length)

RSA Signatures

- An RSA signature scheme consists of 3 algorithms:

  - (PK,SK) <— KeyGen(1^n)
    - (N,e,d) <— GenRSA(1^n)
    - Set PK = (N,e), SK = (N,d)
  - σ <— Sign$_{SK}$($m \in Z_N^*$)
    - Compute $\sigma = m^d \bmod N$
  - {"accept","reject"} <— Verify$_{PK}$(m,σ)
    - Accept if $m \stackrel{?}{=} \sigma^e \bmod N$

Works, since $\sigma^e \bmod N = m^{e.d \bmod \phi(n)} \bmod N = m^1 \bmod N = m$

Some RSA attacks

- RSA sigs. aren't EUF-CMA secure
- No-message attack:
  - Find a $\sigma \in Z_N^*$
  - Compute $m = \sigma^e \bmod N$; (m,σ) is valid forgery
- Malleability attack:
  - A has message $m \in Z_N^*$
  - A picks $m_1, m_2 \in Z_N^*$, s.t., $m = m_1 \bullet m_2 \bmod N$, gets $\sigma_1, \sigma_2 \in Z_N^*$
  - A outputs $(\sigma_1 \bullet \sigma_2)^e \bmod N$, which is a valid forgery[1]

1: $(\sigma_1 \bullet \sigma_2)^e \bmod N = (m_1^d \bullet m_2^d)^e \bmod N = m_1 \bullet m_2 = m$

A just outputs $(\sigma_1 \bullet \sigma_2)^e \bmod N$ and exits. Verifying $(\sigma_1 \bullet \sigma_2)^e \bmod N \stackrel{?}{=} m$ is done by challenger

RSA-FDH

- RSA Full Domain Hash
- Prevent malleability attacks by hashing messages
- An RSA-FDH signature scheme consists of 3 algorithms:
  - $(PK,SK) \leftarrow KeyGen(1^n)$
    - $(N,e,d) \leftarrow GenRSA(1^n)$
    - Choose $H: \{0,1\}^* \rightarrow Z_N^*$
    - Set $PK = (N,e)$, $SK = (N,d)$
  - $\sigma \leftarrow Sign_{SK}(m \in \{0,1\}^*)$
    - Compute $\sigma = H(m)^d \bmod N$
  - $\{\text{"accept"},\text{"reject"}\} \leftarrow Verify_{PK}(m,\sigma)$
    - Accept if $\sigma^e \stackrel{?}{=} H(m) \bmod N$

RSA-FDH

- Practical? Somewhat…
  - RSA PKCS #1 v.2.1 variant of RSA-FDH
  - RSA PKCS #1 salts (randomizes) message, then repeatedly hashes
  - If salt = NULL, RSA PKCS #1 is same as RSA-FDH
- Cannot use regular hash function, e.g., SHA-2, etc.
  - Range of SHA-2, etc. fixed (160 bits, 256 bits,…)
  - H's range needs to cover all of Z_N^*
  - Small-range H practical attacks known…

Identification Scheme

- Identification = establishing identity
- Authentication (signatures): verifying an established identity

- Traditionally used to construct sig. schemes
  - E.g., Amos Fiat—Adi Shamir transform, ‘86
    - F-S transform has problems, but is used nevertheless[1]…
    - Also used in Zero-Knowledge Proof (ZKP) construction

1: See Goldwasser et al. (https://eprint.iacr.org/2003/034.pdf) for details

Identification Protocol

- Interactive, challenge-response protocol
- Played between two players: prover (P), verifier (V)
- Arthur-Merlin 3-round protocol[1]

- General ID protocol:
  - P's PK is published
  - (I,state) <— P(SK) , sends I to V
  - c <— V(PK,1^n), sends c to P
  - r <— P(SK,state,c), sends r to V
  - V(PK,c,r) $\stackrel{?}{=}$ I, then accept P's identity

1: *Round complexity* important measure of efficiency in two/multi-party protocols

Arthur-Merlin Protocols

- Fascinating class of interactive decision problems in complexity theory (see *complexity zoo*[1] for details; there's a *petting zoo* too!)

- Complexity classes: not just P vs. NP: entire hierarchy from A—Z!
  - AM, MA, BPP, PSpace, NISZK, NIPZK, NPSpace,…, ZPP, and many more…
  - 417 and counting!

- But… this isn't a complexity theory class

1: https://complexityzoo.uwaterloo.ca/Complexity_Zoo

Fiat-Shamir Transform

- Provides a way to convert *any* ID scheme into a sig. scheme
  - (PK,SK) <— Gen(1^n)
    - (pk,sk,challengeSet) <— GenID(1^n)
    - Choose H: $\{0,1\}^*$ —> challengeSet
  - σ <— $\mathrm{Sign}_{SK}(m \in \{0,1\}^*)$
    - Compute (I,state) <— P(SK)
    - Compute c <— H(I,m)
    - Compute r = P(SK,state,c)
    - Set $\sigma = (c, r)$
  - {“accept”,”reject”} <— $\mathrm{Verify}_{PK}(m,\sigma)$
    - Compute I = $\mathrm{Verify}_{PK}(c, r)$
    - If H(I, m) $\stackrel{?}{=}$ c, return “accept”

Schnorr Identification Scheme

- (PK,SK) <— Gen(1^n) /* Run by P */
  - (G, q, g)<— GroupGen(1^n) /* log q = n */
  - Pick x <— Z_q, set y = g^x
  - Set PK = (G, q, g, y), SK = x

- P picks k $\in Z_q^*$, set I = g^k, sends I to V
- c $\in Z_q$ <— V(PK,1^n), sends c to P
- P does r = cx + k mod q, sends r to V
- V accepts if $g^r \bullet y^{-c} \stackrel{?}{=} I$

Schnorr Signature Scheme

- Fiat-Shamir(Schnorr ID scheme) —> Schnorr sig. scheme
- (PK,SK) <— Gen(1^n)
  - (G, q, g)<— GroupGen(1^n) /* log q = n */
  - Pick x <— Z_q, set y = g^x, pick H: $\{0,1\}^*$ —> Z_q
  - Set PK = (G, q, g, y), SK = x
- σ <— $Sign_{SK}(m \in \{0,1\}^*)$
  - Picks $k \in Z_q$, set $I = g^k$,
  - Compute c <— H(I,m)
  - Compute r = cx + k mod q
  - Set σ = (c, r)

Schnorr Signature Scheme

- {"accept","reject"} <— $\text{Verify}_{PK}(m, \sigma = (c,r))$
  - Compute $I = g^r \bullet y^{-c}$
  - If $H(I, m) \stackrel{?}{=} c$, return "accept"

Digital Signature Algorithm (DSA)

- Used in Digital Signature Standard (DSS) issued by NIST
- F (ID scheme) —> Sig. scheme; F – transformation function
- But F slightly different from F-S transform

DSA's ID scheme

Prover Alice. $SK_A = x$, $PK_A = (G,g,q,y = g^x)$

Verifier Bob

1. Picks $k \in Z_q^*$, computes $I = g^k$

2. Send I

3. Pick $\alpha, r \in Z_q$

4. Send challenge = α, r

5. Compute $s = (k^{-1} \bullet (\alpha + x \bullet r) \bmod q)$

6. Send response, s

7. Accept Alice's identity as valid iff: $((s \neq 0)$ AND $(g^{\alpha s^{-1}} \bullet y^{rs^{-1}} \stackrel{?}{=} I))$

DSA's ID scheme

- Correctness:
  - s = 0 with negl. probability — this only happens when s = -xr mod q
  - If $s \neq 0$, Step 7 works correctly
  - How?
  - Since $g^{\alpha s^{-1}} \bullet y^{rs^{-1}}$

$$= g^{\alpha s^{-1}} \bullet g^{xrs^{-1}}$$
$$= g^{(\alpha+xr)s^{-1}}$$
$$= g^{(\alpha+xr)\bullet k\bullet(\alpha+xr)^{-1}}$$
$$= I$$

- Fun, isn't it!?
- Next, transform DSA's ID scheme into a signature scheme

DSA

Prover Alice. $SK_A = x$, $PK_A = (G,g,q,y = g^x)$ Verifier Bob

1. Setup a $H:\{0,1\}^* \longrightarrow Z_q$, $F:G \longrightarrow Z_q$
2. Pick an $m \in \{0,1\}^*$ to sign
3. $\sigma \longleftarrow Sign_{SKA}(m)$
   3.1. Pick $k \in Z_q^*$, set $r = F(g^k)$
   3.2. Compute $s = (k^{-1} \bullet (H(m) + x \bullet r) \bmod q)$
   3.3. If $r, s = 0$, start over with fresh k
4. Output $\sigma = (r,s)$

5. Send signature, $\sigma = (r,s)$ $\longrightarrow$

6. Accept iff: $((r,s \neq 0)$ AND $F(g^{H(m) \bullet s^{-1}} \bullet y^{r \bullet s^{-1}}) \stackrel{?}{=} r)$

DSA

- Use good PRF for $k \in Z_q^*$ (must be really random)
  - … else, Bob immediately gets $SK_A = x$

- Since $s = k^{-1} \bullet (H(m) + xr) \bmod q$
  - Bob knows $\sigma = (r,s)$, and m
  - G, g, $|G| = q$, of course are public
  - Only 2 unknowns: k, x

DSA

- Dangerous to re-use same k for 2 different signatures (by same Alice)
- $s_1 = k^{-1} \bullet (H(m_1) + x \bullet r) \bmod q$
- $s_2 = k^{-1} \bullet (H(m_2) + x \bullet r) \bmod q$

- Do $s_1 - s_2 = k^{-1} \bullet (H(m_1) - H(m_2)) \bmod q$[1]
- Get k, then easy to get x

- Sony Playstation (PS3) master-key-extraction attack 2010

1: r will repeat, since $r = F(g^k)$

Security?

- But… we've only talked about correctness, what about security?

- Plain RSA — not EUF-CMA secure
- RSA-FDH — proven EUF-CMA secure
  - Sketch: reduction-based proof
  - A — RSA-FDH adversary. Assume A exists, plays EUF-CMA game
  - B — *Factoring, H-collision* adversary, plays factoring game
  - B can, by interacting with A, break *factoring assumption* or find *collision in H*
  - Contradiction. Qed.

Actual proof in a much disputed model — Random Oracle (RO) model, but something better than nothing…

Security?

- Schnorr ID scheme, Schnorr sig. scheme – proven EUF-CMA secure
  - Sketch: reduction-based proof
  - A – Schnorr adversary. Assume A exists, plays EUF-CMA game
  - B – *Discrete Log (DL)* adversary, plays DL game
  - B can, by interacting with A, break DL assumption
  - Contradiction. Qed.

Security?

- DSA? Haha :-)
  - No proof exists[1]
  - Based on DL hardness assumption
  - Intuitively, should be hard to forge, if DL assumption holds
  - But still widely used…
  - No known attacks — if used *sensibly* (see slide 27, 28 for a "non-sensible" use)

1: Mainly because no proofs known for specific choices of "F" in the standard

Lamport's Signature Scheme

- Leslie Lamport, 1979 (of LaTeX fame, among others)
- Scheme *not* based on number-theoretic assumptions!
  - Elegant, very appealing![1]
- Rather, based on hash functions
- "One-time-secure" signature scheme
  - An SK is used to sign only a *single* message

1: Trivia tidbit: Lamport won ACM Turing award, 2013

Lamport Scheme

- Set $SK_A = [x_{1,0} \;\; x_{2,0} \;\; x_{3,0}$
  $x_{1,1} \;\; x_{2,1} \;\; x_{3,1}]$; $x_{i,j} \in \{0,1\}^n$
- Set $PK_A = [y_{1,0} \;\; y_{2,0} \;\; y_{3,0}$
  $y_{1,1} \;\; y_{2,1} \;\; y_{3,1}]$; $y_{i,j} = H(x_{i,j})$
- Consider $m = (m_1 || m_2 || m_3)$ = "011"
- For signing m, release x_{i,m_i} for each bit i of m:
  - So, $\sigma = (x_{1,0}, x_{2,1}, x_{3,1})$

Lamport Scheme

- Verification:
  - Given $\sigma = (x_{1,0}, x_{2,1}, x_{3,1})$, PK_A, and $m = 011$
  - Check if σ is valid

- Accept as valid iff: $H(x_i) \stackrel{?}{=} y_{i,m_i}$; $\forall\ 1 \leq i \leq 3$
  - If $H(x_1) \stackrel{?}{=} y_{1,0}$, and
  - If $H(x_2) \stackrel{?}{=} y_{2,1}$, and
  - If $H(x_3) \stackrel{?}{=} y_{3,1}$

- For successful forgery, A must find H^{-1} of un-used elements in PK

Lamport Scheme

- Easily generalizes to n-bit messages, but high storage overhead
- Use:
  - When traditional public-key crypto sigs. cannot be used
  - Quantum-resistant (potentially)
- Proven secure?
  - Yes, assuming H is one-way
  - Reduction-based proof — works in usual way

Lamport Scheme

- Optimizations:
  - Use a Merkle hash-tree for storing PK – only root-hash need to be published
  - For SK, store a single PRF seed, generate 2n SK components when required, $n = |m|$
  - Winternitz optimization – reduces |PK|, |SK|, but increases computation…

Key Distribution

- One application of signatures — to distribute public keys
- Digital certificate: $Cert_{Alice}$ <— $Sign_{SKCA}(PK_{Alice})$[1]
- Alice sends (PK_{Alice},$Cert_{Alice}$) to Bob; Bob does: {"accept", "reject"} <— $Verify_{PKCA}(Cert_{Alice})$
- Can send over *insecure*, *un-authticated* channel, as long as CA isn't compromised

1: CA: certification authority

PKI

- Public-key Infrastructure (PKI) defines how:
  - CA verifies Alice
  - Bob gets PKCA

- Some PKIs:
  - Single CA
  - Multiple CAs
  - Delegation/Certificate chains
  - Web-of-trust, e.g., PGP

More info about PKI can be found in a network security text: e.g., see Stallings, or Kaufman, Perlman, Speciner

PKI - Single CA

- CA trusted by everyone, accessible to everyone, e.g., govt. agency, dept., company, etc.
- Everyone gets copy of PK_{CA} (securely)
- CA could bundle PK_{CA} with other software
  - E.g., web-browser + PK_{CA}, browser automatically verifies certificates as they come
- CA needs to verify IDs carefully before issuing certificates

PKI- Multiple CAs

- Motivation: obvious
  - Single CA might get corrupted,
  - ….or might have lax verification process (single-factor auth., etc.),
  - ….or might be lax with its own SK_{CA} storage

- Alice gets $Cert_{A1}$, $Cert_{A2}$, $Cert_{A3}$ from CA1, CA2, CA3, gives all to Bob
- Bob decide which to trust: security only as good as least-trusted CA

PKI-Multiple CAs

- OS, browsers pre-configured with multiple CA's PK
- Default: all treated equally trustworthy
- Fine-grained trust settings might help
  - E.g., ($Cert_{CA1}$ AND $Cert_{CA2}$) OR ($Cert_{CA3}$) OR ($Cert_{CA4}$ AND $Cert_{CA5}$ AND $Cert_{CA6}$)

PKI-Delegation

- Intuitive idea:
  - $Cert_{Alice}$ <— $Sign_{SKCharlie}(PK_{Alice})$
  - $Cert_{Bob}$ <— $Sign_{SKAlice}(PK_{Bob})$
  - $Cert_{Denise}$ <— $Sign_{SKBob}(PK_{Denise})$

- Each $Cert_i$ must also include info that issuer authorized to issue $Cert_i$.

- Root CA, 2nd level CAs, 3rd-level CAs, etc.

- More points of attack

PKI-Web-of-trust

- Informal: anyone can be a CA
- Alice has PK_{Bob}, PK_{Denise} — obtains them at a conference
- George has:
  - $Cert_{George} \leftarrow Sign_{SKBob}(PK_{George})$
  - $Cert_{George} \leftarrow Sign_{SKCharlie}(PK_{George})$
  - $Cert_{George} \leftarrow Sign_{SKDenise}(PK_{George})$
  - $Cert_{Geroge} \leftarrow Sign_{SKTrent}(PK_{George})$
- George presents all cert., Alice decides which to trust

Pretty Good Privacy (PGP): http://pgp.mit.edu :database of public keys, certificates

Key Revocation

- Static: certificates expire at fixed time, unless re-issued (with re-validated credentials)

- Dynamic: revocation-on-demand
  - Include unique serial number with each certificate
  - Publish revoked serial numbers on public revocation list (CRL)
  - Verifiers check CRL each time

PKI

- Don't want/like PKI? Too much hassle?
  - Real concern for many applications

- Use "PKI-less" public-key crypto
  - Identity-based Encryption (IBE)
  - Attribute-based crypto (ABE/ABS)
  - Or other signature paradigms (group, mesh, etc.)
  - None come cheap, but…

No crypto comes on the cheap or for free; if your application cannot bear the computational costs, maybe just avoid using crypto…?

Signcryption

- Goal: provides (confidentiality + integrity), even against CCA2 adversaries

- Encrypt-then-authenticate
- …Or, Authenticate-and-encrypt

- Assume underlying encryption algorithm is CCA2-secure, and sig. scheme is EUF-CMA-secure

Signcryption: Attempt 1

Encryption keys: (EK_A, DK_A)
Signing keys: (SK_A, VK_A)

Encryption keys: (EK_B, DK_B)
Signing keys: (SK_B, VK_B)

1. Do $C = EK_B(m)$
2. Send(Alice, C, $SK_A(C)$)
3. Strips off Alice's sig., replaces with (Charlie, C, $SK_C(C)$)
4. Bob won't notice anything amiss

"Encrypt-then-authenticate" approach

Signcryption: Attempt 2

Encryption keys: (EK_A, DK_A)
Signing keys: (SK_A, VK_A)

Encryption keys: (EK_B, DK_B)
Signing keys: (SK_B, VK_B)

1. Do $\sigma = SK_A(m)$
2. Compute $C = EK_B(m||\sigma)$

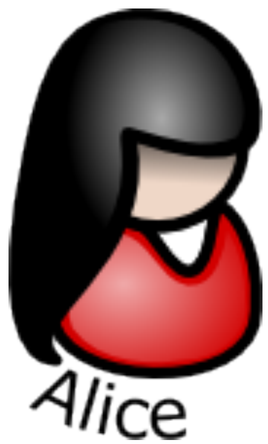
3. Send (Alice, C) $\longrightarrow$

4. Do $(m||\sigma) \leftarrow DK_B(C)$
5. $VK_A(m,\sigma) \stackrel{?}{=}$ "accept"

"Authenticate-then-encrypt" approach

Signcryption: Attempt 2

Encryption keys: (EK_A, DK_A)
Signing keys: (SK_A, VK_A)



Encryption keys: (EK_C, DK_C)
Signing keys: (SK_C, VK_C)

Encryption keys: (EK_B, DK_B)
Signing keys: (SK_B, VK_B)

1. Do $\sigma = SK_A(m)$
2. Compute $C = EK_C(m||\sigma))$
3. Send(Alice,C) $\longrightarrow$
4. Do $(m||\sigma) \leftarrow DK_C(C)$
5. Compute (Alice, $C' = EK_B(m||\sigma))$
6. Send(Alice,C′) $\longrightarrow$
7. Bob'll think (m,σ) came from Alice

"Authenticate-then-encrypt" approach

Signcryption

- Both attempt 1, 2 work
  - Attempt 1 fix: Step 1 – Alice does $C = E_{K_B}(\text{Alice}||m)$
  - Attempt 2 fix: Step 1,2 – Alice does $\sigma = S_{K_A}(\text{Charlie}||m)$, then compute $C = E_{K_C}(\text{Alice}||m||\sigma))$

- Signing – include ID of recipient inside σ
- Encrypting – include ID of sender inside C

Rigorous treatment of signcryption: An, Dodis and Rabin, Eurocrypt'02

Other Sig. Paradigms

- Up until now: traditional public-key signatures (both plain and hash-and-sign versions)

- Others?
  - Group sig.
  - Threshold sig.
  - Attribute-based sig.
  - Ring sig.
  - Mesh sig.

- Pick one: Group sig. (simply because they're widely used, and easy to discuss)

Group Sig.

- Basic idea: group of people, each wants to sign on behalf of group, anonymously

- Parties involved:
  - Group manager: issues SKs to all members, sets GPK
  - Group members: produce sig. verifiable by GPK
- Signatures:
  - Member ID anonymous, but tied in to group
  - Manager can trace signatures

Group Sig.

- A group sig. scheme defined by 4 algorithms:
  - (GPK,GMSK,SK[1..n]) <— GKeyGen(1^k,1^n): randomized
  - σ <— GSign$_{SKi}$($m \in \{0,1\}^*$): randomized
  - {“accept”,”reject”} <— GVerify$_{GPK}$(m,σ): deterministic
  - $\{i,\perp\}$ <— Open(GMSK,m,σ): deterministic

Group Sig. Security

- What “security” are we looking for?
  - Correctness
  - EUF-CMA (unforgeability)
  - Members anonymity
  - Signatures’ traceability to group
  - Unlinkability
  - Exculpability (protection against framing by rogue group members)
  - Collusion-resistance, and …

- Ugh! Can we unify them into a single threat model? Yes!

Group Sig. Security

- Correctness
- Full-anonymity: A shouldn't be able to guess id i, given σ:
    - Even if she knows SK[1..n],
    - Even if she observes results of $\text{Verify}_{GPK}(\bullet,\bullet)$,
    - Even if she observes results of OpenGMSK(•,•)!
- Full-traceability: A cannot create a σ that:
    - Cannot be opened
    - Cannot be traced to a group member
    - Even if GMSK is compromised![1]

1: Models situation where manager's secret key is leaked to A, does *not* model corrupt group manager — corrupt manager can do a lot more!

Group Sig. Security

- The 3 properties *imply* *all* other security properties — unified threat model
    - For static groups
    - If manager is honest-but-clumsy (loses keys)
- *Necessary and sufficient* properties for group sigs.
    - Get all other properties “for free”
- Bellare et al.’03 — seminal paper in group sig.

M. Bellare, D. Micciano, B. Warinschi. Foundations of group signatures. In Proc. of Eurocrypt’03, pp.614—629.

Group Sig.

- Only seen static groups

- Dynamic:
  - Partially dynamic: New users join (“append-only”)
  - Fully dynamic: Users join and leave

- Additional security property: *Forward security*
  - K: ($StartTime_K$, $EndTime_K$)
  - A shouldn’t be able to forge sig. for times > i, if her key revoked at time interval i
  - Subtle point: Nor for *back-dated* times [1..i-1]!

Group Sig.

- What if group manager is corrupt?
  - Will try to issue bad keys, frame members, collude, etc.
  - Actually easy to deal with: ask manager to include a proof of work (POW) with every output

- This, and a lot more: Bellare et al. paper

- Really. go. read. it.