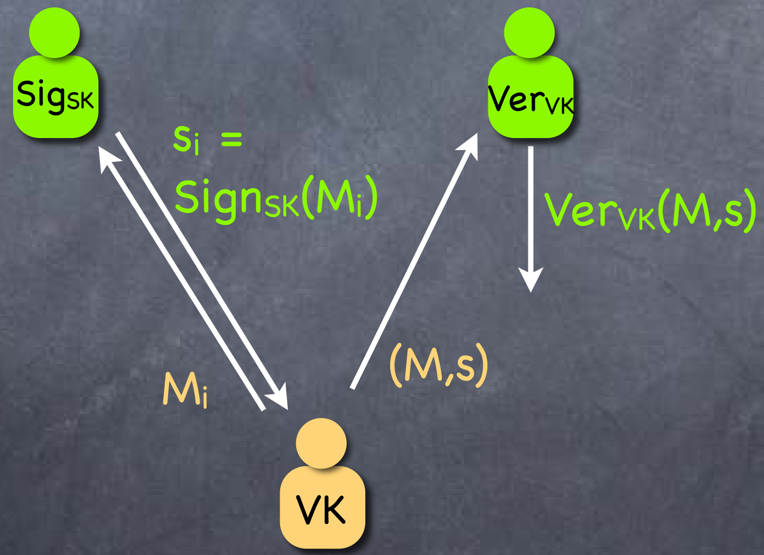
Digital Signatures (ctd.)

Lecture 17

RECALL

Digital Signatures

- Syntax: KeyGen, Sign_{SK} and Verify_{VK}.
  Security: Same experiment as MAC's, but adversary given VK



Advantage = $\Pr[\ \text{Ver}_{VK}(M,s)=1 \text{ and } (M,s) \notin \{(M_i,s_i)\}\ ]$

Weaker variant: Advantage = $\Pr[\ \text{Ver}_{VK}(M,s)=1 \text{ and } M \notin \{M_i\}\ ]$

RECALL

Digital Signatures

- Syntax: KeyGen, $Sign_{SK}$ and $Verify_{VK}$.
  Security: Same experiment as MAC's, but adversary given VK
- Secure digital signatures using OWF, UOWHF and PRF
  - Hence, from OWF alone (more efficiently from OWP)
- More efficient using CRHF instead of UOWHF
- Even more efficient based on (strong) number-theoretic assumptions
  - e.g. Cramer-Shoup Signature based on "Strong RSA assumption"
- Efficient schemes secure in the Random Oracle Model
  - e.g. RSA-PSS in RSA Standard PKCS#1

One-time Digital Signatures

Lamport's One-Time Signature

- Recall One-time MAC to sign a single n bit message
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i_{m_i})_{i=1..n}$
- One-Time Digital Signature: Same signing key and signature, but VK= $(f(r^i_0), f(r^i_1))_{i=1..n}$ where f is a OWF
  - Verification applies f to signature elements and compares with VK
  - Security [Exercise]

$f(r^1_0)$	$f(r^2_0)$	$f(r^3_0)$
$f(r^1_1)$	$f(r^2_1)$	$f(r^3_1)$

r^1_0	r^2_0	r^3_0
r^1_1	r^2_1	r^3_1

Full-Fledged Signatures

- Lamport's scheme is one-time and has a fixed-length message (and SK/VK are much longer than the message)
  - One-time, fixed-length signatures (Lamport)

    $\xrightarrow{\text{"Certificate Tree"}}$ many-time, fixed-length signatures (using PRF)

    $\xrightarrow{\text{Domain-Extension}}$ full-fledged signatures (using UOWHF)
- So full-fledged digital signatures can be entirely based on OWF
- Hash-and-Sign domain extension for signatures
  - Domain extension can be done using CRHF (more efficient) or UOWHF (more secure)

One-Time → Many-Times

- Certificate chain: $VK_1 \to (VK_2, \sigma_2) \to \ldots \to (VK_t, \sigma_t) \to (m,\sigma)$ where σ_i is a signature on VK_i that verifies w.r.t. VK_{i-1}
  - Suppose a "trustworthy" signer only signs the verification key of another "trustworthy" signer. Then, if VK_1 is known to be issued by a trustworthy signer, and all links verified, then the message is signed by a trustworthy signer.
- Certificate tree for one-time → many-times signatures
  - Idea: Each message is signed using a unique VK for that message
    - Verifier can't hold all VKs: A binary tree of VKs, with each leaf designated for a message. Parent VK signs its pair of children VKs (one-time, fixed-length sign). Verifier remembers only root VK. Signer provides a certificate chain to the leaf VK used.
    - Signer can't remember all SKs: Uses a PRF to define the tree (i.e., SK for each node), and remembers only the PRF seed

Domain Extension of Signatures using Hash

- Domain extension using a CRHF (not weak CRHF, unlike for MAC)
  - $\mathrm{Sign}^*_{SK,h}(M) = \mathrm{Sign}_{SK}(h(M))$ where $h \leftarrow \mathcal{H}$ in both SK*,VK*
  - Security: Forgery gives either a hash collision or a forgery for the original (finite domain) signature
  - Formal reduction to a pair of adversaries. Hash adversary sends h it receives as part of VK
- Can use UOWHF, with fresh h every time (included in signature)
  - $\mathrm{Sign}^*_{SK}(M) = (\, h, \mathrm{Sign}_{SK}(h, h(M))\,)$ where $h \leftarrow \mathcal{H}$ picked by signer
  - Security?
  - To use a signature s_i in forgery, need M such that $h(M)=h(M_i)$. But h is picked by signing algorithm after M_i is submitted. Breaks UOWHF security by finding such a collision.
    - In reduction, hash adversary guesses an i where collision occurs and sends h it received as part of signature

More Efficient Signatures

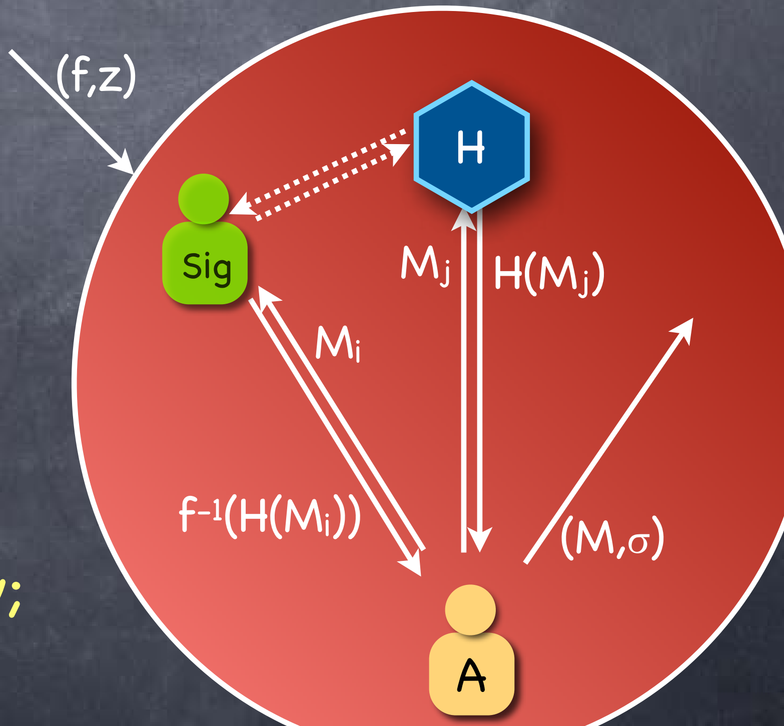
- Diffie-Hellman suggestion (heuristic): Sign(M) = f^{-1}(M) where (SK,VK) = (f^{-1},f), a Trapdoor OWP pair. Verify(M,σ) = 1 iff f(σ)=M.
  - Attack: pick σ, let M=f(σ) (Existential forgery)
- Fix: **Sign(M) = f^{-1}(Hash(M))**
  - Secure? Adversary gets to choose M and hence Hash(M); so signing oracle gives adversary access to f^{-1} oracle. But Trapdoor OWP gives no guarantees when adversary is given f^{-1} oracle.
  - If Hash(.) modeled as a random oracle then adversary can't choose Hash(M), and effectively doesn't have access to f^{-1} oracle. Then indeed secure
  - "Standard schemes" like RSA-PSS are based on this

Proving Security in the RO Model

- To prove: If Trapdoor OWP secure, then $\text{Sign}(M) = f^{-1}(\text{Hash}(M))$ is a secure digital signature in the RO Model, with Hash modelled as a random oracle
  - Intuition: adversary only sees $(x, f^{-1}(x))$ where x is random, which it could have obtained anyway, by picking $f^{-1}(x)$ first
- Modeling as an RO: RO randomly initialized to a random function H from $\{0,1\}^*$ to $\{0,1\}^k$
  - Signer and verifier (and forger) get oracle access to H(.)
  - All probabilities also over the initialization of the RO

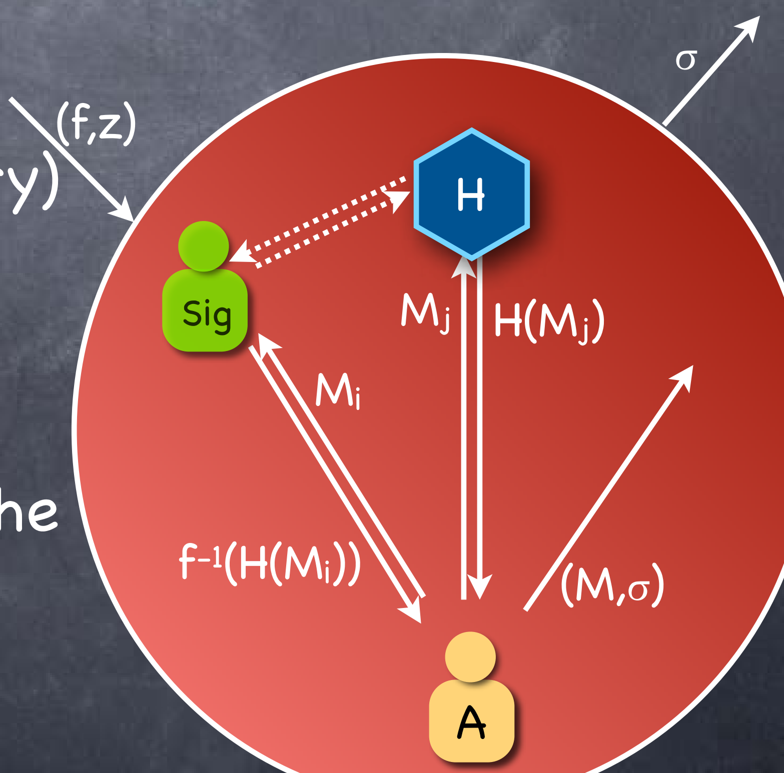
Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from Trapdoor OWP and H an RO), then A* that can break Trapdoor OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.
  - A expects f, access to the RO and a signing oracle f^{-1}(Hash(.)) and outputs (M,σ) as forgery
  - A* can implement RO: a random response to each new query!
  - A* gets f, but doesn't have f^{-1} to sign
    - But x = H(M) is a random value that A* can pick!
    - A* picks H(M) as x=f(y) for random y; then Sign(M) = $f^{-1}(x)$ = y



Proving Security in ROM

- A^* s.t. if A forges signature, then A^* can break Trapdoor OWP
  - A^* implements H and Sign: For each new M queried to H (including by Sign), A^* sets H(M)=f(y) for random y; Sign(M) = y
  - But A^* should force A to invert z
    - For a random (new) query M (say t^{th}) A^* sets H(M)=z
      - Here queries include the "last query" to H, i.e., the one for verifying the forgery (which may or may not be a new query)
  - Given a bound q on the number of queries that A makes to Sign/H, with probability 1/q, A^* would have set H(M)=z, where M is the message in the forgery
    - In that case forgery $\Rightarrow \sigma = f^{-1}(z)$



Schnorr Signature

- Public parameters: (G,g) where G is a prime-order group and g a generator, for which DLA holds, and a random oracle H
  - Or (G,g) can be picked as part of key generation
- Signing Key: $y \in Z_q$ where G is of order q. Verification Key: $Y = g^y$
- $Sign_y(M) = (e,s)$ where $e = H(M||g^r)$ and $s = r-ye$, for a random r
- $Verify_Y(M,(e,s))$: Compute $R = g^s \cdot Y^e$ and check $e = H(M||R)$
- Secure in the Random Oracle model under the Discrete Log Assumption for a group
  - Alternately, under a heuristic model for the group (called the Generic Group Model), but under standard-model assumptions on the hash function