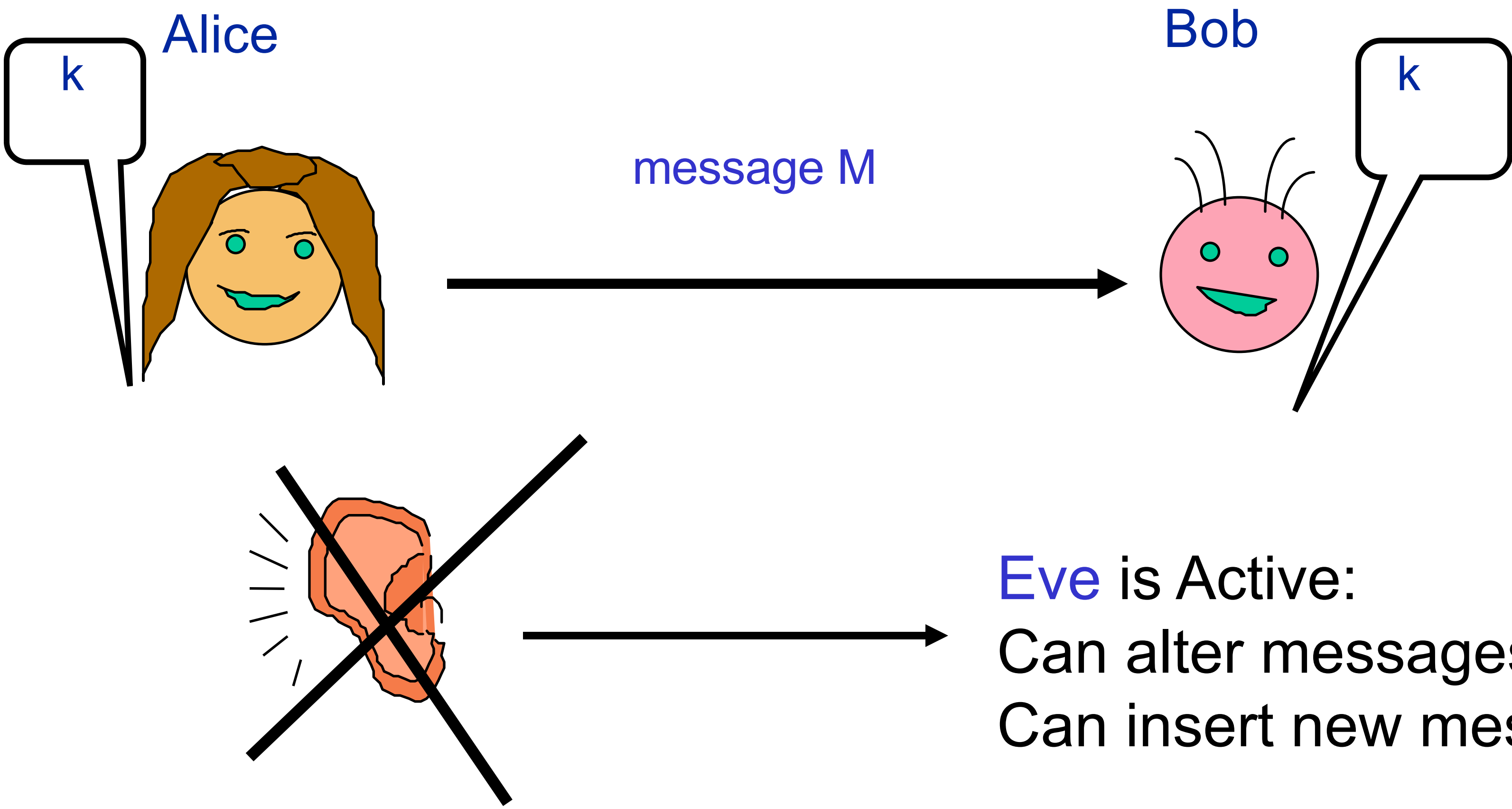
Message Authentication Codes

Digital Signatures

Lecture 11

Shafi Goldwasser

Authentication Problem



Eve is Active:

Can alter messages

Can insert new messages

Authentication Problem

- Secrecy is not the only concern
- Integrity of the message may be even more important for applications. An Active adversary may
  - alter messages in transit
  - inject new messages
  - remove messages

Message Authentication Codes

A way to associate a ***tag*** with each ***message*** which is hard to produce without knowing the secret key

Formal:

A Triplet of algorithms (Gen, MAC, Verify)

- Gen(1^n) produces key $k \in K_n$
- MAC (k,M): on key k and message M, outputs tag t
- Verify(k,M,t) on key k, message M & tag t
  outputs {Accept, Reject} where

Correctness: for all m, Verify(k, m, MAC(k,m)) = Accept

Hard to Forge (needs a definition):

Intuitively, hard to generate new (m, t) s.t.
Verify(k,m,t)=accept

Comments

MAC may be

- Probabilistic: there are may be many tags for the same message (not a requirement for achieving security)
- Deterministic: Verify(k,M,t) simply re-computes $t' = MAC(k,M)$ and compares $t =? t'$

Verify may be

- probabilistic correct with high probability.

Replay: Definition includes only stateless Algorithms, for dealing with replay we may modify this assumption

What is the power of the adversary?

- Can see pairs of (m, MAC(k,m))

- Can access a Verify_k :=Verify(k, ,) oracle
  - Can check if tag are valid for m, tag of its choice
  - Practice: send a (m, tag) & see if accepted or not.

- Can access Mac_k := MAC(k,) oracle
  - Obtain tags for messages of choice

Chosen Message Attack(CMA):Both powers

Who is a successful forger

After attack forger can

- **Total Break:** recover the secret key
- **Universal Break**: generate tags for any message
- **Existential Forgery:** ∃message m for which can generate a tag t s.t. Verify(k,m,t) = accept

Q: Is this too strong?
Why not allow for forging tags for nonsense messages?
A: Definition of `nonsense' is application specific

Security Definition for MAC scheme (Gen, MAC, Verify)

$\forall$adversary A $\exists$neg() s.t. $\forall$n sufficiently large

$$\text{Prob}_{k \in Gen(1^n)}[A^{Verify_k, MAC_k}(1^n)=(m,t) \text{ s.t. } Verify_k(m,t)=Accept \ \& \ m \notin \{m_i \text{ queries by } A^{Verify_k, MAC_k}\}] < neg(n)$$

Can consider adversary A which is:

- Unbounded: information theoretic setting
- Polynomial time in n=|secret key|
- Exact security: **(T,ε) – secure** if for all adversary A who can make T calls to MAC_k succeeds with probability < **ε**

Replay Attack

- **Replay**: sending the exact same (m,t) at a later time
  - Definition of Security Doesn't rule it out
- In practice:
  - Time Stamps appended to messages -- Need Synchronized Clocks
    - Take a Window to Allow for clock drifts
  - Sequence Numbers appended to messages
    - This requires stateful MAC and Verify algorithms, would need to modify our definition accordingly

Beware: Privacy and Authentication Two Entirely Different Goals

- False intuition: $E_k(m)$ garbles m so why not use MAC(k,m) = E(k,m) ?

- Even though adversary can't learn m from E(k,m) may still be able to modify (m, E(k,m)) to (m', E(m')) s.t. Verify(k,m',E(k,m'))= Y

- One Time PAD provides a trivial example: can generate valid tags for new messages from old (message, tag) pairs.

PSRF imply Secure MAC schemes for Fixed Size Messages

Theorem:

- Let $F_n=\{f_k: \{0,1\}^B \to \{0,1\}^B\}$ PRF family
- Then there exist a secure message authentication scheme for B- bit messages

$$MAC(k,M) = f_k(M)$$

MAC for Long Messages?

Let PSRF $F=\{F_n\}$, $F_n=\{f_k\}$, f_k: $\{0,1\}^B \rightarrow \{0,1\}^B$

- $MAC0\ (k,M^0 \ldots M^l) = f_k(M^0 \otimes M^2 \ldots \otimes M^l)$
  - Existential forgery as long as $\otimes M = \otimes M'$
- $MAC1\ (k,M^0 \ldots M^l) = \oplus_i f_k(M^i)$ for $|M^i|=B$, use padding for messages which are not multiples of B in length
  - Order-of-blocks forgery
- $MAC2\ (k,M^0 \ldots M^l) = \otimes_i (f_k(<i>.M^i))$ for $|M^i|=B/2$
  - Cut and paste attack on 3 messages

Randomize

- Let PSRF $F=\{F_n\}$, $F_n=\{f_k\}$, f_k: $\{0,1\}^B \rightarrow \{0,1\}^B$

- Choose random $r \in \{0,1\}^{B/2}$, let $|M^i|=B/2$

  XOR-MAC $(M^0 \ldots M^l)$ =

  $$[r,\ f_k(<0>:r) \otimes f_k(<1>:M^1) \otimes \ldots\ f_k(<l>:M^l)]$$

  - pad if message length not multiple of B/2
  - Make r long enough so chance of collision with r by another r' is small.

- Challenge: **prove** this works if F PSRF
- "Bellare, Guerin, Rogaway, "XOR MACS"

Hash-then-Sign

- Let $H:\{0,1\}^* \Rightarrow \{0,1\}^n$ be a collision resistant hash function
  - Function which can be evaluated by all
  - Function which compresses arbitrary length messages to n bit strings
  - Hard to find collisions

    $\forall$ppt A, Prob[A(H)=(x.x') s.t. H(x)=H(x')] < neg(n)

- Not known to follow from one-way permutation
- Known constructions from DLP, Factoring, LWE
- Real life implementations: MD5, SHA-1

Hash-then-Sign

- Let $H:\{0,1\}^*\Rightarrow\{0,1\}^n$ be a collision resistant hash function

- Gen: On input 1^n choose PSRF f_k in F_n
- MAC: On f_k and message m output $t= f_k(H(m))$
- Verify: On input f_k , m and t
  - Compute H(m)
  - if $f_k(H(m))=t$ output Accept else Reject

Note: forge either by breaking f_k
or by finding collisions: i.e m' s.t. H(m)=H(m')
for m previously signed

Digital Signatures

Wish List for Handwritten Signatures

- Associate documents with a signer (individual)
- To verify need to compare against other signatures
- Signatures are legally binding
- Should be hard to forge
- Should be hard to change the document once its signed

Wish List for **Digital** Signatures

- Associate documents with a signer (user in a computer network)
- Computationally easy to verify **by everyone**, but hard to forge for all except for the legal signer
- Non-refutable: if Alice signs a document, then she cannot deny it.
  - In particular, should not be able to change document once it is signed

$\Rightarrow$Legally binding

Digital Signatures vs. MAC

- Digital signatures are the public-key (or asymmetric) analogue of MACs
  - Publicly Verifiable
  - Transferable: can show the signature to a third party who can verify that the signature is valid
  - Can not be refuted: if Alice signs a document for Bob, she cannot deny it.

Digital Signature: Definition

A digital signature is a triplet of PPT algorithms

- $G(1^k)$ outputs pair (s,v) where s is referred to as the signing key and v the verifying key. [(s,v) ε $G(1^k)$]
- Sign (s,m) on signing key s and message m, outputs s referred to as the digital signature of m [sig ε Sign(s,m)]
- Verify(v,m,sig) on verifying key v, message m, and sig outputs accept or reject s.t.

  Verify(v,m,sig) =accept (sig is a valid signature of m)
  =reject (sig in invalid signature of m).

Correctness: Verify(v,m,s)=accept if sig ε Sign(s,m) where (s,v) in $G(1^k)$

Security : to be defined

Power of the adversary/forger?

Forger can:

- **Key Only Attack:** see only the public verifying key
- **Known Message Attack:** see the public key and pairs of (m, Sign(s,m)) for m signed in the past
- **Chosen Message Attack:** Forger can request to see signatures of messages of his choice
- **Adaptively Chosen Message Attack:** Forger can request to see signatures of messages of his choice which may be chosen in a way dependent on previous signatures seen

Successful Forgery

- **Total Break**: Forger recovers the secret signing key
- **Universal Forgery**: for any message m Forger can come up with a string sig which will be accepted as a valid signature of m by the Verify algorithm
- **Existential Break:** There exist some message for which the forger can produce a valid signature

Security Definition for MAC scheme (G, Sign, Verify)

$\forall$adversary A $\exists$neg() s.t. $\forall$n sufficiently large

Prob$_{(s,v)\in G(1^n)}$ [$A^{Sign_k}(v)$=(m,t) s.t Verify(v,m,t)=Accept &
m $\notin$ {m_i queries by A to oracle Sig(s,)] <neg(n)

Can consider adversary A which is:

- Polynomial time in n=|secret key|
- Exact security: **(T,ε) – secure** if for all adversary A who can make T calls to Sign(s,) succeeds with probability < **ε**

Remarks

- Could it be made any Stronger ?
  - How?
  - do not allow forger to produce a different signature for the same message signed in the past

Digital Signatures: Primary Usages

- **Authenticity** of documents: A digital signature provides a way for each user in a network to sign messages so that signatures can later be verified by anyone.

- **Integrity** of signed documents: Anyone can verify that the content of a document that have been signed has not been altered.

- **Certificates**

Certificates

- If the directory of public keys is accessed over the network, one needs to protect the users from fraudulent public keys.
- Certificates -- a user's public key digitally signed by the public key directory manager (as a trusted party) is one solution to this problem.
- Each user can transmit this certificate along with his public key with any message he signs *removing the need for a central directory.*
- The only thing that need be trusted is that the directory manager's public key is authentic.

Public-Key Infrastructure (PKI)

- Trusted root authority (VeriSign, IBM, United Nations)
  - Everyone must know the verification key of root authority
- Root authority can sign certificates
- Certificates identify others, including other authorities
- Leads to certificate chains

Digital Signatures: Trapdoor Function Model

- Diffie Hellman 76 proposal in our notation is: given a trapdoor collection of functions F define (Gen,Sign,Verify) as follows
- Gen: On input security parameter 1^n, pick a function f in F_n and its associated trapdoor t. Make the signing key t and the verifying key is f.
- Sign(t,m) $= f^{-1}(m)$
- Verify(f,m,sig) = accept if f(sig) =m

  and reject otherwise

Why does it work?

Since $f(sig) = f(f^{-1}(m)) = m$ when $sig = f^{-1}(m)$ as computed by the legal signing algorithm.

Existential Forgery

Even though F is a collection of trapdoor functions, the scheme is trivial to "existentially forge" under a "key only" attack as follows

On public key f in F,

Adversary A chooses at random x in the domain of f and sets message=f(x),
signature=x.

How about signing single bit messages ?

Instantiation: The RSA Digital Signature Scheme

The first example of a digital signature scheme was proposed by the RSA in 77.

- Key Generation: choose $n=pq$ and e, d s.t. $ed=1 \bmod \phi(n)$ Set (n,e) the public verifying key and d the private signing key.
- Sign(d,m)

  Set $sig = m^d \bmod n$ to be the signature of m
- Verify ((n,e), sig,m):

  output 1 if and only if $(sig)^e \bmod n = m$.

Why ? $sig=m^d \bmod n$ implies $sig^e \bmod n =(m^{de})=$

$m^{ed \bmod \phi(n)} = m \bmod n$

Security of RSA signatures

Claim: RSA is existentially forgeable under a Key only attack.

Proof: Let pk=(n,e) and sk –d s.t. ed=1 mod phi(n).

Simply choose x in Z_n* at random, and set m=x^e mod n,

and sig=x, then V((n,e),sig, m)=accept. Namely, x is a legal signature of m.

Claim: RSA is universally forgeable under chosen message attack(CMA)

Proof: Suppose interested in forging the signature of m. Choose a random r in Z_n*. Let m_1=r and m_2=m/r mod n. Get signatures s_1 =$(m_1)^d$ mod n, s_2=$(m_2)^d$ mod n of m_1, m_2 from S (during the CMA). Now, it is easy to compute the signature of m,

set s=s_1*s_2 mod n=$(m_1)^d$ $(m_2)^d$ mod n=$(m_1*m_2)^d$ mod n =m^d mod n.

Hash-then-Sign RSA

- Hash-then-Sign paradigm
- Generation: PK = ((n, e), H), SK = (p,q)
- Signing: On input signing key d and message m output $s = H(m)^d$ mod n.
- Verifying: On input (n,e), s, and m,
  - Compute H(m)
  - if s^e mod n = H(m) output 1 (accept signature)

Note: can try to forge either by breaking RSA or by looking for collisions, i.e m and m' H(m)=H(m')

Note: has the added advantage of handling long messages "for free"

Security of hashed RSA

- Theorem: if **H is a random oracle**, then Hashed RSA signatures is existentially secure against chosen message attack under the RSA assumption.
- Variants of hashed RSA have been standardized, and are used in practice
- Problem: H is not really a random oracle is

In Practice: PSS0- RSA

- Hash-then-Sign probabilistic paradigm
- Generation: PK = ((n, e), H), SK = (p,q)
- Signing: On input signing key d and message m output (s,r)
  - where σ = H(r|| m)^d mod n
  - r is randomly chosen each time, |r| = |m|
- Verifying: On input (n,e), (s,r) and m, output 1 if and only if s^e mod n = H(r||m)

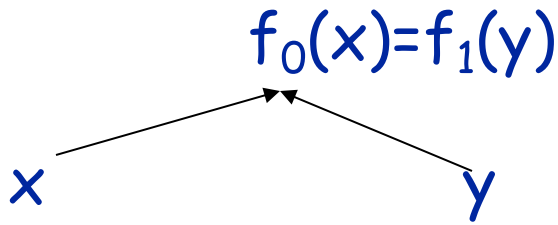
Important Remark

- Diffie Hellman in their work linked the tasks of public-key encryption and digital signatures.
- They observed that for pair of (E,D) (from public-key encryption) you can use s=D(m) as a signature and E(s) =? M as the verifying algorithm.
- This of course fails when E is a probabilistic scheme and is not true in general for any encryption scheme.
- We explicitly separate the two tasks to achieve greater security.

Next time Show:
How to Sign **any message**
Securely from any one-way
functions

Start with signing 1 message

How to Sign a Bit: Claw-Free Functions



Let (f_0,f_1) be a pair of trapdoor functions which are "claw-free", i.e. its hard to find x,y s.t. $f_0(x)=f_1(y)$

Let

verifying key vk	signing key sk
z, f_0, f_1	f^{-1}, f^{-1}_1 (the corresponding trapdoor information)

- To sign b, output $=f_b^{-1}(z)$
- To verify that σ is a valid signature of b, check if $f_b(\sigma)=z$ for z in public file.