Digital Signatures - Idea
Public key encryption idea

Digital Signatures - Idea
Digital signature idea

Digital Signatures - Idea
**Digital Signatures - How it Works**

Signature scheme consists of three algorithms:
- **Generate keypair**: Given keylength (security param) gives (PU, PR)
- **Sign**: Takes message M and PR, and produces signature sig
- **Verify**: Takes M, PU, and sig, and outputs true (verified) or false

Like public key encryption, sign/verify operations are slow!
- So don’t run entire (possibly long) message through functions
- First hash, then sign $H(M)$

Is this combination secure? Yes! Why: Assume adversary knows valid sigs $(M_1, \text{sig}_1), (M_2, \text{sig}_2), \ldots, (M_n, \text{sig}_n)$ and can find a forgery $(M, \text{sig})$.
- If $H(M) = H(M_i)$ for some $M_i \rightarrow$ found a collision in $H$, should be impossible!
- If $H(M) \neq H(M_i)$ for all $M_i \rightarrow$ then $(H(M), \text{sig})$ is a forger for sig scheme

**Digital Signatures - Security Model**

```
A(PU)
// Arbitrary precomputation
while (not done):
    m = // compute query message
    s = S(m)
    Known = Known U (m,s)
    // More computing
    (m', s') = // compute claimed forgery
    Return (m', s')
Adversary wins if there is no pair (m', s) in Known and Verify(m', s) = true

Note:
- Adversary picks oracle query messages, and can adapt as it learns
  - That makes this an ‘adaptive chosen message’ attack
- Any valid signature wins - only restriction is that m’ hasn’t been queried
  - That makes this an ‘existential forgery attack’

Security is Existentially Unforgeable under Adaptive Chosen Message Attack (EUF-CMA)
```

**ElGamal**

As in Diffie-Hellman, let $p$ be a prime and $g$ be a primitive root

**Key Generation**
1. Pick random $PR \in [2, \ldots, p-1]$
2. Compute $PU = g^{PR} \mod p$
3. Private (signing) key is $PR$; Public (verification) key is $PU$

**Signing a message M**
1. Pick random $k \in [2, \ldots, p-1]$ that is relative prime to $(p-1)$
2. Compute $r = g^k \mod p$
3. Compute $k^{-1} \mod (p-1)$
4. Compute $s = k^{-1}(H(M) - PRr) \mod (p-1)$
5. Signature is the pair $(r,s)$

**Verifying a signature (r,s) on message M**
1. Check if $g^s = PU^r \cdot r^M \mod p$ [accept if true, reject if false]
**ElGamal**

As in Diffie-Hellman, let \( p \) be a prime and \( g \) be a primitive root

Key Generation
1. Pick random \( PR \in \{2, \ldots, p-1\} \)
2. Compute \( PU = g^{PR} \mod p \)
3. Private (signing) key is \( PR \); Public (verification) key is \( PU \)

Signing a message \( M \)
1. Pick random \( k \in \{2, \ldots, p-1\} \) that is relative prime to \((p-1)\)
2. Compute \( r = g^k \mod p \)
3. Compute \( k^{-1} \mod (p-1) \)
4. Compute \( s = k^{-1} (H(M) - PR^{r}) \mod (p-1) \)
5. Signature is the pair \((r,s)\)

Verifying a signature \((r,s)\) on message \( M \):
1. Check if \( g^{sr} \equiv PU^{-1} \mod p \) [accept if true, reject if false]

---

**Why does this work for valid sigs?**

**Important math fact:** If \( x \equiv y \pmod{p-1} \) then \( a^x \equiv a^y \pmod{p} \).

**Proof:** If \( x \equiv y \pmod{p-1} \) then there exists a \( k \) such that \( x \equiv k(p-1)+y \). Then \( a^x \equiv a^{k(p-1)+y} \equiv (a^{p-1})^k a^y \). By Fermat’s Little Theorem, we know that \( a^{p-1} \equiv 1 \pmod{p} \), so \( (a^{p-1})^k a^y \equiv a^y \). Therefore \( a^x \equiv a^y \pmod{p} \).

What this means: To simplify \( a^{\text{exponent}} \), can simplify formula \( \pmod{p-1} \).

Applying this to ElGamal formulas:

\[
PU = g^{PR} \mod p \\
PR^r = g^{k(H(M) - PR^{r})} \mod (p-1) \\
\]

Consider \( PU^r \equiv g^{k(H(M) - PR^{r})} \pmod{p} \), and simplify exponent \( \pmod{p-1} \):

\[
PR^r + k^s = PR^r + k^s (H(M) - PR^r) = PR^r + H(M) - PR^r = H(M) \pmod{(p-1)} \\
\]

Therefore, \( PU^r \equiv g^{H(M)} \pmod{p} \)

---

**DSA - Digital Signature Algorithm**

**Compared to ElGamal**

**ElGamal**

Let \( q \equiv p-1 \)

Key Generation
1. Pick random \( PR \in \{2, \ldots, q\} \)
2. Compute \( PU = g^{PR} \mod p \)
3. Private key is \( PR \); Public key is \( PU \)

Signing a message \( M \)
1. Pick random \( k \in \{2, \ldots, q\} \) with \( g^{bk} \equiv 1 \pmod{p} \)
2. Compute \( r = g^k \mod p \)
3. Compute \( k^{-1} \mod q \)
4. Compute \( s = k^{-1} (H(M) - PR^{r}) \mod q \)
5. Signature is the pair \((r,s)\)

Verifying signature \((r,s)\) on message \( M \):
1. Check if \( g^{sr} \equiv PU^{-1} \mod p \)
DSA - The Digital Signature Algorithm

One component of NIST’s Digital Signature Standard (DSS)
- DSS was adopted in 1993
- DSA dates back to 1991
- One goal: Only support integrity - not confidentiality
  - Why? Export restrictions!
  - Alternative signature scheme: RSA - also an encryption algorithm

Key and Parameter Sizes:
- ElGamal is similar to Diffie-Hellman modulus size (\(N\) = number of bits)
  - 1024-bit \(p\) was OK in 1990s - now suggest 2048-bit or 3072-bit
  - Signature two \(N\)-bit values (e.g., two 1024-bit values)
- DSA uses a computationally-hard subgroup
  - In 1990’s \(q\) was 160 bits (matching SHA1!)
  - Signature was then two 160-bit values (more compact than ElGamal)
  - Now suggest \(q\) being 256 bits

Reminder - RSA Algorithm

From Public Key Encryption chapter

Key Generation:
- Pick two large primes \(p\) and \(q\)
- Calculate \(n = p \cdot q\) and \(\phi(n) = (p-1)(q-1)\)
- Pick a random \(e\) such that \(\gcd(e, \phi(n)) = 1\)
- Compute \(d = e^{-1} \text{ (mod } \phi(n))\) [Use extended GCD algorithm]
- Public key is \(PU = (n, e)\); Private key is \(PR = (n, d)\)

Encryption of message \(M \in \{0,..,n-1\}\):
- \(E(PU, M) = M^e \mod n\)

Decryption of ciphertext \(C \in \{0,..,n-1\}\):
- \(D(PR, C) = C^d \mod n\)

Correctness - easy when \(\gcd(M, n) = 1\):
- \(D(PE(E(PU, M)) = (M^e)^d \mod n = M^{ed} \mod n = M^{(e\phi(n)+1)} \mod n = (M^{\phi(n)})^M \mod n = M\)

Also works when \(\gcd(M, n) \neq 1\), but slightly harder to show...

RSA Algorithm for Signatures

“Textbook algorithm” - not how it’s really done

Key Generation:
- Pick two large primes \(p\) and \(q\)
- Calculate \(n = p \cdot q\) and \(\phi(n) = (p-1)(q-1)\)
- Pick a random \(v\) such that \(\gcd(v, \phi(n)) = 1\)
- Compute \(s = v^{-1} \text{ (mod } \phi(n))\) [Use extended GCD algorithm]
- Public key is \(PU = (n, v)\); Private key is \(PR = (n, s)\)

Signing message \(M \in \{0,..,n-1\}\):
- \(Sign(PR, M) = M^s \mod n\)

Verification of signature \(\sigma \in \{0,..,n-1\}\):
- \(Verify(PU, M, \sigma): \text{ Check if } M = \sigma^v \mod n\)
RSA-PSS (Probabilistic Signature Scheme)
How it’s really done - with padding (similar to OAEP for encryption)

Invented (and proved secure) by Bellare and Rogaway
- Also inventors of OAEP and HMAC

Forging sigs w/ “textbook RSA”
- Pick random sig $R$
- Let message $M = R^v \mod N$
- $(M,R)$ is valid sig pair!

Modifying sigs (“blinding”)
- Given $\sigma = M^s \mod N$
- Compute $X = R^v \mod N$
- Let $M' = X^v M \mod N$
- Let $\sigma' = R^v \sigma \mod N$
- Note $(\sigma')^v = R^v \sigma^v = X^v M \equiv M' \pmod N$