Asymmetric Cryptography

Daniel Halperin
Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Class updates

• Remember current events and security reviews are due this Friday

• Lockpicks and now **Fingerprint molds** are available in my office

• **Office hours** today in CSE 210
Class updates (cont.)

- Lab 3 coming soon - Privacy
- Working out the details with the lawyers
- Homework 3 (last homework!) out by Wednesday - Hashing and Asymmetric Cryptography
Some Number Theory Facts

- Euler totient function $\varphi(n)$ where $n \geq 1$ is the number of integers in the $[1,n]$ interval that are relatively prime to $n$
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
- Euler’s theorem: if $a \in \mathbb{Z}_n^*$, then $a^{\varphi(n)} \equiv 1 \pmod{n}$
  - $\mathbb{Z}_n^*$: multiplicative group of integers mod $n$ (integers relatively prime to $n$)
- Special case: Fermat’s Little Theorem
  - if $p$ is prime and gcd($a,p$)=1, then $a^{p-1} \equiv 1 \pmod{p}$
RSA Cryptosystem

[Rivest, Shamir, Adleman 1977]

Key generation:
• Generate large primes p, q
  – Say, 1024 bits each (need primality testing, too)
• Compute n=pq and ϕ(n)=(p-1)(q-1)
• Choose small e, relatively prime to ϕ(n)
  – Typically, e=3 or e=2^{16}+1=65537 (why?)
• Compute unique d such that ed = 1 mod ϕ(n)
• Public key = (e,n); private key = (d,n)

Encryption of m:  c = m^e \mod n
• Modular exponentiation by repeated squaring

Decryption of c:  c^d \mod n = (m^e)^d \mod n = m
Why RSA Decryption Works

\( e \cdot d = 1 \mod \varphi(n) \), thus \( e \cdot d = 1 + k \cdot \varphi(n) \) for some \( k \)

Can rewrite: \( e \cdot d = 1 + k(p-1)(q-1) \)

\( m \) be any integer in \( Z_n \)

If \( \gcd(m, p) = 1 \), then \( m^{ed} = m \mod p \)
- By Fermat’s Little Theorem, \( m^{p-1} = 1 \mod p \)
- Raise both sides to the power \( k(q-1) \) and multiply by \( m \)
- \( m^{1+k(p-1)(q-1)} = m \mod p \), thus \( m^{ed} = m \mod p \)
- By the same argument, \( m^{ed} = m \mod q \)

Since \( p \) and \( q \) are distinct primes and \( p \cdot q = n \),

\( m^{ed} = m \mod n \) (using the Chinese Remainder Theorem)

True for all \( m \) in \( Z_n \), not just \( m \) in \( Z_n^* \)
Why Is RSA Secure?

- **RSA problem**: given $n = pq$, $e$ such that $\gcd(e, (p-1)(q-1)) = 1$ and $c$, find $m$ such that $m^e = c \mod n$
  - i.e., recover $m$ from ciphertext $c$ and public key $(n, e)$ by taking $e^{th}$ root of $c$
  - There is no known efficient algorithm for doing this

- **Factoring problem**: given positive integer $n$, find primes $p_1$, ..., $p_k$ such that $n = p_1^{e_1}p_2^{e_2}...p_k^{e_k}$

- If factoring is easy, then RSA problem is easy, but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring $n$
Caveats

- $e = 3$ is a common exponent
  - If $m < n^{1/3}$, then $c = m^3 < n$ and can just take the cube root of $c$ to recover $m$
    - Even problems if “pad” $m$ in some ways [Hastad]
  - Let $c_i = m^3 \mod n_i$ - same message is encrypted to three people
    - Adversary can compute $m^3 \mod n_1n_2n_3$ (using CRT)
    - Then take ordinary cube root to recover $m$

- Don’t use RSA directly for privacy!
Integrity in RSA Encryption

- Plain RSA does **not** provide integrity
  - Given encryptions of $m_1$ and $m_2$, attacker can create encryption of $m_1 \cdot m_2$
    - $(m_1^e) \cdot (m_2^e) \mod n = (m_1 \cdot m_2)^e \mod n$
  - Attacker can convert $m$ into $m^k$ without decrypting
    - $(m_1^e)^k \mod n = (m^k)^e \mod n$

- In practice, OAEP is used: instead of encrypting $M$, encrypt $M \oplus G(r) \; ; r \oplus H(M \oplus G(r))$
  - $r$ is random and fresh, $G$ and $H$ are hash functions
  - Resulting encryption is **plaintext-aware**: infeasible to compute a valid encryption without knowing plaintext
    - ... if hash functions are “good” and RSA problem is hard
OAEP (image from PKCS #1 v2.1)

The diagram illustrates the OAEP (Optimal Asymmetric Encryption Padding) process as described in PKCS #1 v2.1. The process involves the following steps:

1. **DB** is the plaintext message block, which is divided into three parts: IHash, PS, and M.
2. A seed is generated and used in conjunction with a function MGF to produce a masked message (M ⊕ G(r)).
3. The masked message is then further processed to produce the final ciphertext EM = maskedSeed || maskedDB.

The equation $r \oplus H(M \oplus G(r))$ is used to generate the seed.
Today So Far

- Defined RSA primitives
- Encryption and Decryption
- Underlying number theory
- Practical concerns, some mis-uses
- OAEP
Digital Signatures: Basic Idea

**Given:** Everybody knows Bob’s **public key**
Only Bob knows the corresponding **private key**

**Goal:** Bob sends a “digitally signed” message
1. To compute a signature, must know the private key
2. To verify a signature, enough to know the public key
RSA Signatures

- Public key is \((n,e)\), private key is \(d\)
- To sign message \(m\): \(s = m^d \mod n\)
  - Signing and decryption are the same **underlying** operation in RSA
  - It’s infeasible to compute \(s\) on \(m\) if you don’t know \(d\)
- To verify signature \(s\) on message \(m\):
  \[s^e \mod n = (m^d)^e \mod n = m\]
  - Just like encryption
  - Anyone who knows \(n\) and \(e\) (public key) can verify signatures produced with \(d\) (private key)

- In practice, also need padding & hashing
  - Standard padding/hashing schemes exist for RSA signatures
Often people think: Encryption and decryption are inverses.

That’s a common view

- True for the RSA **primitive (underlying component)**

But not one we’ll take

- To really use RSA, we need padding
- And there are many other decryption methods
Digital Signature Standard (DSS)

- U.S. government standard (1991-94)
  - Modification of the ElGamal signature scheme (1985)

- Key generation:
  - Generate large primes \( p, q \) such that \( q \) divides \( p-1 \)
    \[-2^{159} < q < 2^{160}, 2^{511+64t} < p < 2^{512+64t}\] where \( 0 \leq t \leq 8 \)
  - Select \( h \in \mathbb{Z}_p^* \) and compute \( g = h^{(p-1)/q} \mod p \)
  - Select random \( x \) such \( 1 \leq x \leq q-1 \), compute \( y = g^x \mod p \)

- Public key: \((p, q, g, y=g^x \mod p)\), private key: \( x \)

- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract \( x \) (private key) from \( g^x \mod p \) (public key)
DSS: Signing a Message (Skim)

Message
Hash function (SHA-1)
Random secret between 0 and q

Compute $r = (g^k \mod p) \mod q$

Private key

Compute $s = k^{-1} \cdot (H(M) + x \cdot r) \mod q$

$(r,s)$ is the signature on $M$
DSS: Verifying a Signature (Skim)

- Compute $w = s'^{-1} \mod q$
- Compute $(g^{H(M')}w \cdot y^{r'w} \mod p) \mod q$
- If they match, signature is valid