Asymmetric Encryption

With material from Jonathan Katz, David Brumley, and Dave Levin
• Warmup activity
• Overview of asymmetric-key crypto
• Intuition for El Gamal and RSA
  • And intuition for attacks
• Digital signatures / authenticity
Public-Key Crypto
• Recall our three goals:
  • Confidentiality
  • Integrity
  • Authenticity
Recall: Drawbacks of symmetric crypto

- How to securely exchange keys?
- Hard to scale
- Limited authenticity / non-repudiation

We will use asymmetric crypto to mitigate these drawbacks!
High-level idea

• Generate a pair of keys
  • One for encryption, one for decryption

• Make encryption key public!
  • On your website, in the New York Times
  • Anyone can send you a private message

• Secret key is the *trapdoor*
Warmup Activity
Public key example map
Message = 66
Private key map

Minimum dominating set = NP hard
Message = 66
Your turn! Public map
private map
Notes on this example

• Finding the (a) private map is very hard
  • Minimum dominating set (NP)
  • For a sufficiently large map

• But, can solve as a system of linear equations

• So, this is **not secure**
  • But it is kind of a fun illustration
Asymmetric crypto

• $k_e \neq k_d$
• $k_d =$ **private** key, $k_e =$ **public** key
  • Bob computes both, gives public key to Alice
• Alice sends a message to Bob: $c = E(m, k_e)$
• Bob can decrypt it: $m = D(m, k_d)$
• Anyone can send, only **Bob** can read!
Asymm. Cryptosystem: Definition

- Three polynomial-time algorithms:
  - KeyGen: Returns $k_p$ (public) and $k_s$ (secret)
  - $E(k_p, m)$: Encrypts $m$ with $k_p$, returns $c$ in $C$
    - Must be randomized (why?)
  - $D(k_s, c)$: Decrypts $c$ with $k_s$, returns $m$ in $M$
    - Or error

- Correctness condition:
  - For all pairs $(k_p, k_s)$: $D(k_s, E(k_p, m)) = m$
Pros and Cons

- Scales well — everyone makes one key pair
  - Not \( n \) keys
- No direct setup comms between Alice and Bob
- Asymmetric is \textit{much, much slower}
- Asymmetric is easier to attack
  - Requires stronger assumptions
The authenticity problem

• In symmetric, we needed an authentic, private channel to exchange keys
  • Diffie-Hellman let us relax to authentic only
  • Public-key also requires authentic channel

• Who posted that ad in the NY Times?
  • Much more on this later
In practice: Hybrid

- Bob generates key pair and publishes $k_p$
- Alice generates new symmetric key $k_{AB}$
- Alice -> Bob: $c_1 = E(k_p, (Alice || k_{AB}))$
- Alice -> Bob: $c_2 = E(k_{AB}, \text{message})$
- Arbitrary-length messages, efficiently
  - Keep $k_{AB}$ as a session key
Intuition for algorithms
El Gamal (simplified)

- Similar to Diffie-Hellman
  - Public key: prime $p$, generator $g$, $h = g^x$
  - Private key: $x$

- Encryption: Sender chooses $y$
  - $c_1 = g^y$, $c_2 = m^y h^y$

- Decryption: $m = c_2 / c_1^x$

- Security equivalent to D-H hardness
A teeny bit of number theory

• $N = pq$, where $p$ and $q$ are distinct primes

• $\phi(N) = (p-1)(q-1)$
  • Easy to compute if you know $p$ and $q$; hard if not

• $a^b \mod N = a^{b \mod \phi(N)} \mod N$
  • Take my word or take 456

• $\mathbb{Z}_M^*$: integers relatively prime to $M$
  • Have no common denominators except 1
Building to RSA (simplified)

- Choose \( e \) relatively prime to \( \phi(N) \)
  - You can do do mod arithmetic
- Choose \( d \) s.t. \( e \cdot d \mod \phi(N) = 1 \)
  - Easy if you know \( \phi(N) \); else hard
    - By extension, easy if you know \( p \) and \( q \)
- Public key = \( (e, N) \); Private key = \( d \)
Textbook RSA

- Encrypt: \( c = m^e \mod N \)
- Decrypt: \( m = c^d \mod N \)
- Why does this work? \( m^{ed} = m^1 = m \)
Textbook RSA: NOT Secure

- Deterministic
-Leaks info about plaintext
  - In practice: Preprocess message before applying RSA permutation
   - Randomized padding, hash permutations
PKCS #1 v1.5

• You need 1024 total bits

• Pad message: $c = (r || m)^e \mod N$
  • $r$ is (mostly) a random number

• Check padding on decryption to detect error
Is RSA hard?

• Easy to compute $m$ when we know $d$ (of course)
  • But what about if we don’t?

• Challenge: Compute $x$ given $c = m^e \mod N$
  • Easiest known way: Factor $N$ into $p$ and $q$
    • Believed (not proven) nothing easier
  • Factoring $N$ is believed hard (but not proven)
How hard is hard?

- Best current algorithms to factor $N=pq$
  - $p$ and $q$ equal-length
  - runs in $\approx \exp(|N|^{1/3})$

- Currently $|N| \sim 1024$ for OK security
  - $\sim 2048$ to be sure
How hard is hard?

• World record: RSA-768 (232 digits)
  • Two years, hundreds of machines
  • Equivalent to 2000 single-core years!

• Factoring 1024-bit integer
  • About 1000 times harder
  • …. Possible this decade?
Implementation attacks

- Timing and power:
  - How long / how much to compute $c^d \bmod N$

- Bad randomness:
  - $p$ and $q$ can’t be predictably generated
  - If $N = pq$ and $N’ = pq’$, both are broken

- Bad padding / malleability
Malleability

- Given $c$ (m unknown), can construct $c'$ that will decrypt to a related message $m'$
- Recall CBC attack last time
CBC is not CCA-secure

Challenge:
Choose $b = x$ or $y$ at uniform random

$m_x$ and $m_y$
$|m_x| = |m_y| = 1$ blk
$c = E(k, m_b) = IV || c[0]$
$c' = (IV \ xor \ 1) || c[0]$
$m' = D(k, c') = m_b \ xor \ 1$

Decryption oracle

Uh oh.
Malleability

- Given $c$ (m unknown), can construct $c'$ that will decrypt to a related message $m'$
  - Recall CBC attack last time
  - CBC, CTR are malleable; auth. encr. is not!

- Basic El Gamal and basic RSA are malleable
  - CCA-safe variations exist
Bleichenbacher attack

- Insecure padding, malleability
  - Return error if padding not formatted correctly
- Allows gradual CCA attack based on error detection
  - Analogous to blind ROP attack?
In practice

• Need CCA security for real applications
• Symmetric: Use authenticated encryption
• Use approved pub key scheme
• Hybrid: Combine!
  • Secure if components are
Digital signatures
Signatures for integrity

• Sign with your private key

• Anyone can verify using public key
  • Assuming private key is secret, only you could have sent the message

• e.g., Sign software patches
  • Public key bundled with initial software
### Signatures vs. MACs

<table>
<thead>
<tr>
<th>Manage one key</th>
<th>Manage n keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign once, verifiable by anyone</td>
<td>Sign separately per verifier</td>
</tr>
<tr>
<td>Public non-repudiation</td>
<td>Nope</td>
</tr>
</tbody>
</table>
Defining a signature scheme

• Keygen: outputs $k_p$ and $k_s$
• $s = S(k_s, m)$
• $V(k_p, m, s)$ outputs true or false
• Correctness:
  • For all pairs $(k_p, k_s)$: $V(k_p, m, S(k_s, m)) = true$
Signature security game

- No existential forgeries (analogous to MAC)

Security IFF \( \Pr[V(k_p, m', s') = 1] \) is very small!
Naive RSA signatures

- Public key \((e, N)\) and private key \((d, N)\)
  - Recall: \(e \cdot d \equiv 1 \pmod{\text{arithmetic}}\)
- \(s = m^d \pmod{N}\)
- Verify whether \(s^e \pmod{N} = m\)
- This is \textit{easily existentially forgeable}
  - Choose \(s\). Calculate \(m\).
RSA signatures (better)

• Send $s = H(m)^d \mod N$ along with $m$
  • Use a good cryptographic hash function $H$

• Recipient calculates digest $g = s^e \mod N$
  • Verify $g == H(m)$

• Why does this fix the problem?
  • You can choose $s'$ and find the matching digest $g'$
  • BUT, preimage resistance means that you can’t pick a message $m'$ s.t. $g == H(m')$

• Variants of this approach are believed secure
  • Assuming RSA is hard
  • Bonus: Handles long messages “for free”