CSE 484: Computer Security and Privacy

Asymmetric Cryptography

Fall 2021

David Kohlbrenner
dkohlbre@cs

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, David Kohlbrenner, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials...
Administrivia

• Lab 1 due on Wednesday
• HW2 next week
Remember our troubles with randomness?
CSPRNGs in practice

• Gather some good entropy (256 bits?)

• Use a block cipher/HMAC/Hash to ‘stretch’ this entropy

• Regularly mix in more entropy!

Check out NIST.SP.800-90
CSPRNG – CTR_DRBG
Why does this work for CSPRNGs?

• To ‘break’ the CSPRNG (that is, predict the next output)
  • Must know state of CSPRNG (key, inputs)
  • Requires breaking the security of the primitive!

• Your CSPRNG is just as secure as the scheme you use the output for!
  • Never ‘loses entropy’, same guarantee as block cipher

• Why mix in new entropy?
  • Can’t hurt, prevents a single bug from breaking the future
CSPRNGs gone bad

- DUAL_EC_DRBG — Dual Elliptic Curve Deterministic Random Bit Generator
  - CSPRNG based on elliptic curve math
  - NSA designed

- DUAL_EC_DRBG has a *backdoor*
  - Special mathematical construction that allows recovery of state!

- Remember DES’s s-boxes?
  - This is the opposite
Session Key Establishment
Modular Arithmetic

- **Given g and prime p**, compute: \( g^1 \mod p, g^2 \mod p, \ldots g^{100} \mod p \)
  - For \( p=11 \), \( g=10 \)
    - \( 10^1 \mod 11 = 10, 10^2 \mod 11 = 1, 10^3 \mod 11 = 10, \ldots \)
    - Produces cyclic group \{10, 1\} (order=2)
  - For \( p=11 \), \( g=7 \)
    - \( 7^1 \mod 11 = 7, 7^2 \mod 11 = 5, 7^3 \mod 11 = 2, \ldots \)
    - Produces cyclic group \{7,5,2,3,10,4,6,9,8,1\} (order = 10)
    - \( g=7 \) is a “generator” of \( \mathbb{Z}_{11}^* \)
Diffie-Hellman Protocol (1976)

Diffie and Hellman Receive 2015 Turing Award

Whitfield Diffie

Martin E. Hellman
Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- **Public info:** $p$ and $g$
  - $p$ is a large prime, $g$ is a **generator** of $\mathbb{Z}_p^*$
    - $\mathbb{Z}_p^* = \{1, 2 \ldots p-1\}$; a $\mathbb{Z}_p^* \ i$ such that $a=g^i \mod p$
    - **Modular arithmetic:** numbers “wrap around” after they reach $p$

```
Alice
Pick secret, random $x$
Compute $k=(g^y)^x=g^{xy} \mod p$

Bob
Pick secret, random $y$
Compute $k=(g^x)^y=g^{xy} \mod p$
```
Example Diffie Hellman Computation
Why is Diffie-Hellman Secure?

• **Discrete Logarithm (DL) problem:**
  given $g^x \mod p$, it’s hard to extract $x$
  • There is no known **efficient** algorithm for doing this
  • This is **not** enough for Diffie-Hellman to be secure!

• **Computational Diffie-Hellman (CDH) problem:**
  given $g^x$ and $g^y$, it’s hard to compute $g^{xy} \mod p$
  • ... unless you know $x$ or $y$, in which case it’s easy

• **Decisional Diffie-Hellman (DDH) problem:**
  given $g^x$ and $g^y$, it’s hard to tell the difference between $g^{xy} \mod p$ and $g^r \mod p$
  where $r$ is random
More on Diffie-Hellman Key Exchange

- **Important Note:**
  - We have discussed discrete logs modulo integers
  - Significant advantages in using **elliptic curve groups**
    - Groups with some similar mathematical properties (i.e., are “groups”) but have better security and performance (size) properties
Diffie-Hellman: Conceptually

Common paint: \( p \) and \( g \)

Secret colors: \( x \) and \( y \)

Send over public transport:
\[ g^x \mod p \]
\[ g^y \mod p \]

Common secret: \( g^{xy} \mod p \)

[from Wikipedia]
Diffie-Hellman Caveats

• Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  • Common recommendation:
    • Choose p=2q+1, where q is also a large prime
    • Choose g that generates a subgroup of order q in \( \mathbb{Z}_p^* \)
    • DDH is hard in this group
  • Eavesdropper can’t tell the difference between the established key and a random value
  • In practice, often hash \( g^{xy} \mod p \), and use the hash as the key
  • Can use the new key for symmetric cryptography

• Diffie-Hellman protocol (by itself) does not provide authentication (against active attackers)
  • Person in the middle attack (also called “man in the middle attack”)
Example from Earlier

- **Given** g and prime p, **compute**: \( g^1 \mod p, g^2 \mod p, \ldots g^{100} \mod p \)
  - For p=11, \( g=10 \)
    - \( 10^1 \mod 11 = 10, 10^2 \mod 11 = 1, 10^3 \mod 11 = 10, \ldots \)
    - Produces cyclic group \{10, 1\} (order=2)
  - For p=11, \( g=7 \)
    - \( 7^1 \mod 11 = 7, 7^2 \mod 11 = 5, 7^3 \mod 11 = 2, \ldots \)
    - Produces cyclic group \{7,5,2,3,10,4,6,9,8,1\} (order = 10)
    - \( g=7 \) is a “generator” of \( \mathbb{Z}_{11}^* \)
  - For p=11, \( g=3 \)
    - \( 3^1 \mod 11 = 3, 3^2 \mod 11 = 9, 3^3 \mod 11 = 5, \ldots \)
    - Produces cyclic group \{3,9,5,4,1\} (order = 5) (5 is a prime)
    - \( g=3 \) generates a group of prime order
Stepping Back: Asymmetric Crypto

• We’ve just seen **session key establishment**
  • Can then use shared key for symmetric crypto

• Next: **public key encryption**
  • For confidentiality

• Then: **digital signatures**
  • For authenticity
Requirements for Public Key Encryption

- **Key generation**: computationally easy to generate a pair (public key $PK$, private key $SK$)

- **Encryption**: given plaintext $M$ and public key $PK$, easy to compute ciphertext $C=E_{PK}(M)$

- **Decryption**: given ciphertext $C=E_{PK}(M)$ and private key $SK$, easy to compute plaintext $M$
  - Infeasible to learn anything about $M$ from $C$ without $SK$
  - Trapdoor function: $\text{Decrypt}(SK, Encrypt(PK, M)) = M$
Some Number Theory Facts

• Euler totient function $\varphi(n)$ ($n \geq 1$) 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
  • Easy to compute for primes: $\varphi(p) = p-1$
  • Note that $\varphi(ab) = \varphi(a) \varphi(b)$ if a & b are relatively prime
RSA Cryptosystem  [Rivest, Shamir, Adleman 1977]

• Key generation:
  • Generate large primes p, q
    • Say, 2048 bits each (need primality testing, too)
  • Compute n=pq and $\varphi(n)=(p-1)(q-1)$
  • Choose small e, relatively prime to $\varphi(n)$
    • Typically, $e=3$ or $e=2^{16}+1=65537$
  • Compute unique d such that $ed \equiv 1 \mod \varphi(n)$
    • Modular inverse: $d \equiv e^{-1} \mod \varphi(n)$
    • Public key = (e,n); private key = (d,n)

• Encryption of m: $c = m^e \mod n$
• Decryption of c: $c^d \mod n = (m^e)^d \mod n = m$
Why is RSA Secure?

• **RSA problem:** given $c$, $n=pq$, and $e$ such that $\gcd(e, \varphi(n))=1$, find $m$ such that $m^e=c \mod n$
  - In other words, recover $m$ from ciphertext $c$ and public key $(n,e)$ by taking $e$th root of $c$ modulo $n$
  - There is no known efficient algorithm for doing this *without* knowing $p$ and $q$

• **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 *(knowing factors means you can compute $d = \text{inverse of } e \mod (p-1)(q-1)$)*
  - It may be possible to break RSA without factoring $n$ -- but if it is, we don’t know how
RSA Encryption Caveats

• Encrypted message needs to be interpreted as an integer less than n
• Don’t use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow
• Plain RSA also does **not** provide integrity
  • Can tamper with encrypted messages

In practice, OAEP is used: instead of encrypting M, encrypt
M ⊕ G(r) || r ⊕ H(M ⊕ G(r))
• r is random and fresh, G and H are hash functions
RSA OAEP

\[ M \oplus G(r) \ || \ r \oplus H(M \oplus G(r)) \]
Review: RSA Cryptosystem

[Rivest, Shamir, Adleman 1977]

- **Key generation:**
  - Generate large primes $p$, $q$
    - Say, 2048 bits each (need primality testing, too)
  - Compute $n = pq$ and $\varphi(n) = (p-1)(q-1)$
  - Choose small $e$, relatively prime to $\varphi(n)$
    - Typically, $e=3$ or $e=2^{16}+1=65537$
  - Compute unique $d$ such that $ed \equiv 1 \pmod{\varphi(n)}$
    - Modular inverse: $d \equiv e^{-1} \pmod{\varphi(n)}$
    - Public key = $(e, n)$; private key = $(d, n)$

- **Encryption of $m$:** $c = m^e \mod n$
- **Decryption of $c$:** $c^d \mod n = (m^e)^d \mod n = m$

How to compute?
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, only the public key is needed
RSA Signatures

• Public key is \((n,e)\), private key is \((n,d)\)
• To sign message \(m\): \(s = m^d \mod n\)
  • Signing & decryption are 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 \(m\):
  verify that \(s^e \mod n = (m^d)^e \mod n = m\)
  • Just like encryption (for RSA primitive)
  • Anyone who knows \(n\) and \(e\) (public key) can verify signatures produced with \(d\) (private key)
• In practice, also need padding & hashing
  • Without padding and hashing: Consider multiplying two signatures together
  • Standard padding/hashing schemes exist for RSA signatures