CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography: Asymmetric Cryptography (finish)

Spring 2017

Franziska (Franzi) Roesner
franzi@cs.washington.edu

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

• Lab #1 due Friday
• Coming up
  – Today: finish crypto
  – Wednesday: tech policy (Emily McReynolds)
  – Friday: systems security (Xi Wang)
  – Then: web security (meets crypto)
• Homework #2 on crypto out tomorrow (due 5/5)
• My office hours are canceled this week
• In-class worksheets
  – No late worksheets (extra credit reading opportunities if you’re concerned)
  – We’ll drop 3/10 worksheets
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 (subgroup of) $\mathbb{Z}_p^*$
    
    • $\mathbb{Z}_p^* = \{1, 2 \ldots, p-1\}$; \( \forall a \in \mathbb{Z}_p^* \ \exists \ i \) such that \( a = g^i \mod p \)
    
    • Modular arithmetic: numbers “wrap around” after they reach p

Pick secret, random X

---

Pick secret, random Y

---

Alice

\[ g^x \mod p \]

Bob

\[ g^y \mod p \]

Compute \( k = (g^y)^x = g^{xy} \mod p \)

Compute \( k = (g^x)^y = g^{xy} \mod p \)
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 and Hellman Receive 2015 Turing Award

Whitfield Diffie

Martin E. Hellman
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
Properties of Diffie-Hellman

• Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  – Eavesdropper can’t tell the difference between the established key and a random value
  – Can use the new key for symmetric cryptography

• Diffie-Hellman protocol (by itself) does not provide authentication
  – Man in the middle attack
Choosing p

In practice, we choose very large primes (~600 digits) of the form:

\[ q = 2p + 1 \]

(\text{where } p \text{ is prime})
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 $\varphi(ab) = \varphi(a) \varphi(b)$
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 $\varphi(n) = (p-1)(q-1)$
  – Choose small $e$, relatively prime to $\varphi(n)$
    • Typically, $e=3$ or $e=2^{16}+1=65537$
  – Compute unique $d$ such that $ed = 1 \mod \varphi(n)$
    • Modular inverse: $d = e^{-1} \mod \varphi(n)$
  – 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

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

- 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 $M \oplus G(r); r \oplus H(M \oplus G(r))$
  – r is random and fresh, G and H are hash functions
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
  - Standard padding/hashing schemes exist for RSA signatures
DSS Signatures

• Digital Signature Standard (DSS)

• 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)
Advantages of Public Key Crypto

• No shared secrets
  – For neither encryption nor authentication
  – This is pretty magical!
  – Can use this for key establishment
Disadvantages of Public Key Crypto

• Calculations are 2-3 orders of magnitude slower
  – Modular exponentiation is an expensive computation
  – Typical usage: use public-key crypto to bootstrap symmetric key

• Keys are longer
  – 1024+ bits (RSA) rather than 128 bits (AES)

• Relies on unproven number-theoretic assumptions
  – What if factoring is easy?
  – (Of course, symmetric crypto also rests on unproven assumptions…)

• How do you authenticate a key?
Cryptography Summary

• Goal: Privacy
  – Symmetric keys:
    • One-time pad, Stream ciphers
    • Block ciphers (e.g., DES, AES) → modes: EBC, CBC, CTR
  – Public key crypto (e.g., Diffie-Hellman, RSA)

• Goal: Integrity
  – MACs, often using hash functions (e.g., MD5, SHA-256)

• Goal: Privacy and Integrity
  – Encrypt-then-MAC

• Goal: Authenticity (and Integrity)
  – Digital signatures (e.g., RSA, DSS)
Problem: How does Alice know that the public key she received is really Bob’s public key?
Threat: Man-In-The-Middle (MITM)
You encounter this every day...

SSL/TLS: Encryption & authentication for connections

Next time:
How do you know it’s really Google’s key?
What is SSL/TLS exactly?