CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography
[Finish Hash Functions; Start Asymmetric Cryptography]

Spring 2019

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 ...
Admin

• Lab 1 due Monday

• Coming up
  – Monday: Adversarial ML (Ivan Evtimov)
  – Today/Wednesday: Finish crypto
  – Friday: start web security!
Common Hash Functions

- MD5 – Don’t Use!
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm)
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Theoretically broken 2005; practical attack 2017!
- SHA-256, SHA-512, SHA-224, SHA-384
- SHA-3: standard released by NIST in August 2015
Google just cracked one of the building blocks of web encryption (but don’t worry)

It’s all over for SHA-1

by Russell Brandom | @russellbrandom | Feb 23, 2017, 11:49am EST

https://shattered.io
Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.

**MAC: message authentication code**
(sometimes called a “tag”)

Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.
HMAC

- Construct MAC from a cryptographic hash function
  - Invented by Bellare, Canetti, and Krawczyk (1996)
  - Used in SSL/TLS, mandatory for IPsec

- Why not encryption?
  - Hashing is faster than block ciphers in software
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption
Authenticated Encryption

• What if we want both privacy and integrity?
• Natural approach: combine encryption scheme and a MAC.
• But be careful!
  – Obvious approach: Encrypt-and-MAC
  – Problem: MAC is deterministic! same plaintext $\rightarrow$ same MAC
Authenticated Encryption

• Instead: Encrypt \textit{then} MAC.

• (Not as good: MAC-then-Encrypt)
Stepping Back:
Flavors of Cryptography

• Symmetric cryptography
  – Both communicating parties have access to a shared random string $K$, called the key.

• Asymmetric cryptography
  – Each party creates a public key $pk$ and a secret key $sk$. 
Symmetric Setting

Both communicating parties have access to a shared random string $K$, called the key.
Asymmetric Setting

Each party creates a public key $pk$ and a secret key $sk$.

Alice
$pk_A, sk_A$

Bob
$pk_B, sk_B$

Adversary
$pk_A, pk_B, sk_A, sk_B$
Public Key Crypto: Basic Problem

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

Goals: 1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself
Applications of Public Key Crypto

• Encryption for confidentiality
  – Anyone can encrypt a message
    • With symmetric crypto, must know secret key to encrypt
  – Only someone who knows private key can decrypt
  – Key management is simpler (or at least different)
    • Secret is stored only at one site: good for open environments

• Digital signatures for authentication
  – Can “sign” a message with your private key

• Session key establishment
  – Exchange messages to create a secret session key
  – Then switch to symmetric cryptography (why?)
Session Key Establishment
Modular Arithmetic

- Refresher in section yesterday
- Given g and prime p, compute:
  \( g^1 \mod p, g^{100} \mod p, \ldots \ g^{100} \mod p \)
  - 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 \( Z_{11}^* \)
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 \(\mathbb{Z}_p^*\)
    - \(\mathbb{Z}_p^* = \{1, 2 \cdots p-1\}; \quad \forall a \in \mathbb{Z}_p^* \exists i \text{ such that } a = g^i \text{ mod } p\)
    - **Modular arithmetic:** numbers “wrap around” after they reach \(p\)

Alice

Pick secret, random \(X\)

Bob

Pick secret, random \(Y\)

\(g^x \text{ mod } p\)

\(g^y \text{ mod } p\)

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

Compute \(k = (g^x)^y = g^{xy} \text{ mod } p\)
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
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]
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
  – Common recommendation:
    • Choose $p=2q+1$, where $q$ is also a large prime
    • Choose $g$ that generates a subgroup of order $q$ in $\mathbb{Z}_p^*$
  – Eavesdropper can’t tell the difference between the established key and a random value
  – 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
  – Party in the middle attack (often called “man in the middle attack”)

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 and Hellman Receive 2015 Turing Award

Whitfield Diffie

Martin E. Hellman
Public Key Encryption
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:** $\text{Decrypt}(SK,\text{Encrypt}(PK,M))=M$
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 $ed \equiv 1 \mod \varphi(n)$
    • 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 RSA Decryption Works (FYI)

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

Let \( m \) be any integer in \( \mathbb{Z}_n^* \) (not all of \( \mathbb{Z}_n \))

\[ c^d \mod n = (m^e)^d \mod n = m^{1+k \cdot \varphi(n)} \mod n \]
\[ = (m \mod n) \ast (m^{k \cdot \varphi(n)} \mod n) \]

Recall: Euler’s theorem: if \( a \in \mathbb{Z}_n^* \), then \( a^{\varphi(n)} = 1 \mod n \)

\[ c^d \mod n = (m \mod n) \ast (1 \mod n) \]
\[ = m \mod n \]

Proof omitted: True for all \( m \) in \( \mathbb{Z}_n \), not just \( m \) in \( \mathbb{Z}_n^* \)
Why RSA Decryption Works (FYI)

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

- Recall $n=pq$ and $\varphi(n)=(p-1)(q-1)$ and $ed \equiv 1 \mod \varphi(n)$

- **Chinese Remainder Theorem**: To show $m^{ed} \mod n \equiv m \mod n$, sufficient to show:
  - $m^{ed} \mod p \equiv m \mod p$
  - $m^{ed} \mod q \equiv m \mod q$

- If $m \equiv 0 \mod p \rightarrow m^{ed} \equiv 0 \mod p$

- Else $m^{ed} = m^{ed-1}m = m^{k(q-1)(p-1)}m = m^{h(p-1)}m$ for some $k$, and $h=k(q-1)$. Why? Recall how $d$ was chosen and the definition of mod.

- **Fermat Little Theorem**: $m^{(p-1)h}m \equiv 1^hm \mod p \equiv m \mod p$
Why is RSA Secure?

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

- **Factoring problem:** given positive integer $n$, find primes $p_1$, ..., $p_k$ 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 $M \oplus G(r) \oplus H(M \oplus G(r))$
  – $r$ 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)

• Again: We’ve discussed discrete logs modulo integers; significant advantages to using elliptic curve groups instead.
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)