Hash Functions,
Public-Key Encryption
CMSC 23200/33250, Autumn 2018, Lecture 6

David Cash
University of Chicago
Plan

1. A few points about hash functions
2. Introducing Public-Key Encryption
3. Math for RSA
4. Security properties of RSA
Assignment 1 is Online and Due Next Wednesday

1. Start early. You can get bogged down in low-level bugs with bits or Python quirks.
2. Please report any “500 Internal Server” Errors privately on Piazza - We will fix them to throw useful error messages.
Hash Functions

Definition: A hash function is a deterministic function $H$ that reduces arbitrary strings to fixed-length outputs.

Some security goals:
- collision resistance: can’t find $M \neq M'$ such that $H(M) = H(M')$
- preimage resistance: given $H(M)$, can’t find $M$
- second-preimage resistance: given $H(M)$, can’t find $M'$ s.t. $H(M') = H(M)$

Note: Very different from hashes used in data structures!

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Output Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>$m = 128$ bits</td>
</tr>
<tr>
<td>SHA-1</td>
<td>$m = 160$ bits</td>
</tr>
<tr>
<td>SHA-256</td>
<td>$m = 256$ bits</td>
</tr>
<tr>
<td>SHA-512</td>
<td>$m = 512$ bits</td>
</tr>
<tr>
<td>SHA-3</td>
<td>$m \geq 224$ bits</td>
</tr>
</tbody>
</table>
Hash Functions are not MACs

Both map long inputs to short outputs… But a hash function does not take a key.

**Intuition**: a MAC is like a hash function, that only the holders of key can evaluate.
Hash Function Security History

- Can always find a collision in $2^{m/2}$ time ($\ll 2^m$ time). “Birthday Attack”
- MD5 (1992) was broken in 2004 - can now find collisions very quickly.
- SHA-1 (1995) was broken in 2017 - A big computer can find collisions
- SHA-256/SHA-512 (2001) are not broken
- SHA-3 (2015) is new and not broken

MD5( d131dd02c5e6e4693d9a0698aff95c 2fcab58712467eab4004583eb8fb7f89 55ad3406f4b320283e4488832571415a 085125e8f7c92g9fd71dbf28037c5b d8823e3156348f5bae6d4ac436c919c6 dd5e2b487daa03fd02396306d248cda0 e99f33420f577ee8ce54b67080a80d1e c69821bcb6a8839396f9652b6ff72a70 )

= MD5( d131dd02c5e6e4693d9a0698aff95c 2fcab50712467eab4004583eb8fb7f89 55ad3406f4b320283e4488832571415a 085125e8f7c92g9fd71dbf28037c5b d8823e3156348f5bae6d4ac436c919c6 dd5e2b487daa03fd02396306d248cda0 e99f33420f577ee8ce54b67080280d1e c69821bcb6a8839396f9652b6ff72a70 )

Xiaoyun Wang (Tsinghua University), 2004
- Broken with clever techniques
- Compare to DES (broken b/c key too short)
Why are collisions bad?

The binary should hash to 3477a3498234f

MD5(100 001) = 3477a3498234f

Hashes to 3477a3498234f, I accept.

MD5(100 001) = 3477a3498234f
MACs from Hash Functions

**Goal:** Build a secure MAC out of a good hash function.

Common construction: $\text{MAC}(K, M) = H(K \ || \ M)$

- Totally insecure if $H = \text{MD5, SHA1, SHA-256, SHA-512}$ (Assignment 2)
- Is secure with SHA-3

Upshot: Use HMAC and avoid various issues.

Later: Hash functions and certificates
Basic question: If two people are talking in the presence of an eavesdropper, and they don’t have pre-shared a key, is there any way they can send private messages?
Switching Gears: Public-Key Encryption

**Basic question:** If two people are talking in the presence of an eavesdropper, and they don’t have pre-shared a key, is there any way they can send private messages?

Diffie and Hellman in 1976: **Yes!**

*Turing Award, 2015, + Million Dollars*

Rivest, Shamir, Adleman in 1978: **Yes, differently!**

*Turing Award, 2002, + no money*

Cocks, Ellis, Williamson in 1969, at GCHQ: **Yes, we know about both…**

*Pat on the back?*
Switching Gears: Public-Key Encryption

**Basic question:** If two people are talking in the presence of an eavesdropper, and they don’t have pre-shared a key, is there any way they can send private messages?

Formally impossible (in some sense): No difference between receiver and adversary.
Switching Gears: Public-Key Encryption

**Basic question:** If two people are talking in the presence of an eavesdropper, and they don’t have pre-shared a key, is there any way they can send private messages?

Message $M$

$R \leftarrow \text{rand}()$

$\langle \text{some bits} \rangle$

$\langle \text{some bits} \rangle$

$\langle \text{some bits} \rangle$

$R' \leftarrow \text{rand}()$

Receive $M$

$M$?

 Doesn’t know $R, R'$,

 Can’t “try them all” (too many)
Switching Gears: Public-Key Encryption

**Definition.** A public-key encryption scheme consists of three algorithms $Kg$, $Enc$, and $Dec$

- **Key generation algorithm $Kg$,** takes no input and outputs a (random) public-key/secret key pair $(PK, SK)$

- **Encryption algorithm $Enc$,** takes input the public key $PK$ and the plaintext $M$, outputs ciphertext $C \leftarrow Enc(PK, M)$

- **Decryption algorithm $Dec$,** is such that $Dec(SK, Enc(PK, M)) = M$
Public-Key Encryption in Action

PK = public key
known to everyone

SK = secret key
known by Receiver only

Kg

PK, SK

PK

M

C = Enc(PK, M)

SK

M

PK
All known Public-Key Encryption uses…

\[ N = pq \]
Some RSA Math

RSA setup

p and q be large prime numbers (e.g. around $2^{2048}$)
N = pq
N is called the **modulus**

- $p=7$, $q=11$ gives $N=77$
- $p=17$, $q=61$ gives $N=1037$
Modular Arithmetic: Two sets

\[ \mathbb{Z}_N = \{0, 1, \ldots, N - 1\} \]
\[ \mathbb{Z}_N^* = \{i : \text{gcd}(i, N) = 1\} \quad (\mathbb{Z}_N^* \not\subset \mathbb{Z}_N) \]

\text{gcd} = "greatest common divisor"

Examples:
\[ \mathbb{Z}_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \]
\[ \mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\} \]

Definition: \( \phi(N) = |\mathbb{Z}_N^*| \)

\( \phi(13) = 12 \quad \phi(15) = 8 \)
Modular Arithmetic

Definition

\[ x \mod N \] means the remainder when \( x \) is divided by \( N \).

\[ \mathbb{Z}^*_15 = \{1,2,4,7,8,11,13,14\} \]

\[ 2 \times 4 = 8 \mod 15 \quad 13 \times 8 = 14 \mod 15 \]

Theorem:

\[ \mathbb{Z}^*_N \] is “closed under multiplication modulo \( N \)”. 
RSA “Trapdoor Function”

**Lemma:** Suppose $e, d \in \mathbb{Z}^*_{\phi(N)}$ satisfy $ed = 1 \mod \phi(N)$. Then for any $x \in \mathbb{Z}_N$ we have that

$$(x^e)^d = x^{ed} = x \mod N$$

**Example:** $N = 15$, $\phi(N) = 8$, $e = 3$, $d = 3$

The satisfy condition in lemma: $ed = 3 \cdot 3 = 9 = 1 \mod 8$

So “powering by 3” always un-does itself.

$$(5^3)^3 = 5^9 = 1953125 = 5 \mod 15$$

Usually $e$ and $d$ are different.
RSA “Trapdoor Function”

Easy given $N, e, x$

$x \mod N$ $\rightarrow$ $y = x^e \mod N$

Hard given $N, e, y$

Finding “e-th roots modulo N” is hard.
Contrast is usual arithmetic, where finding roots is easy.
RSA “Trapdoor Function”

\[ PK = (N, e) \quad SK = (N, d) \quad \text{where} \quad N = pq, \quad ed = 1 \mod \phi(N) \]

\[ \begin{align*}
\text{Enc}((N, e), M) &= M^e \mod N \\
\text{Dec}((N, d), C) &= C^d \mod N
\end{align*} \]

Messages and ciphertexts are in \( \mathbb{Z}_N^* \)

Setting up RSA:
- Need two large random primes
- Have to pick e and then find d
- Don’t worry about how exactly
Encryption with the RSA Trapdoor Function?

- Several problems
  - Encryption of 1 is 1
  - $e=3$ is popular. Encryption of 2 is 8… (no wrapping mod N)
  - RSA Trapdoor Function is deterministic

**Solution**: Pad input $M$ using random (structured) bits.
- Serves purpose of padding and nonce/IV randomization
PKCS#1 v1.5 RSA Encryption

\( N \): \( n \)-byte long integer.
Want to encrypt \( m \)-byte messages.

Enc((\( N, e \)),M):
1. \( \text{pad} \leftarrow (n-m-3) \) random non-zero bytes.
2. \( X \leftarrow 00\|02\|\text{pad}\|00\|M \)
3. Output \( X \mod N \)

Dec((\( N, d \)),M):
1. \( X \leftarrow C^d \mod N \)
2. Parse \( X = aa\|bb\|\text{rest} \)
3. If \( aa \neq 00 \) or \( bb \neq 02 \) or \( 00 \notin \text{rest} \):
   Output \text{ERROR}
4. Parse \( \text{rest} = \text{pad}\|00\|M \)
5. Return \( M \)

Warning: Broken
Bleichenbacher’s Padding Oracle Attack (1998)

PK = (N, e)

Want to decrypt C

C’

ACCEPT or REJECT

System (e.g. webserver)
SK = (N, d)

Infer something about (C’)^d mod N

Dec((N,d), M):
1. X ← C^d mod N
2. Parse X = aa||bb||rest
3. If aa≠00 or bb≠02 or 00∉rest:
   Output ERROR
4. Parse rest = pad||00||M
5. Return M

Originally needed millions of C’. Best currently about 10,000.
Better Padding: RSA-OAEP

RSA-OAEP [Bellare and Rogaway, ‘94] prevents padding-oracle attacks with better padding using a hash function.

(Then apply RSA trapdoor function.)