CSE 484: Computer Security and Privacy

Cryptography 4

Spring 2024

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...
Logistics

• Lab 1b due Wednesday
• Homework 2 will go out today

• Lab 1a grades coming out ~today
  • 1a sploits are posted
  • 1a writeups out soon

• Things not going well? Please reach out to us ASAP!
Defining the strength of a scheme

- **Effective Key Strength**
  - Amount of ‘work’ the adversary needs to do
- **DES**: 56-bits
  - $2^{56}$ encryptions to try ‘all keys’
- **2DES**: 57-bits
  - $2 \times 2^{56}$ encryptions = $2^{57}$
- **3DES**: 112-bits (or sometimes 80-bits)
  - Meet-in-the-middle + more work = $2^{112}$ (for 3 keys, e.g. K1, K2, K3)
  - Various attacks = $2^{80}$ (for 2 keys, e.g. K1, K2, K1)
Standard Block Ciphers

• **DES: Data Encryption Standard**
  - Feistel structure: builds invertible function using non-invertible ones
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

• **AES: Advanced Encryption Standard**
  - New federal standard as of 2001
    - NIST: National Institute of Standards & Technology
  - Based on the Rijndael algorithm
    - Selected via an open process
  - 128-bit blocks, keys can be 128, 192 or 256 bits
Encrypting a Large Message

• So, we’ve got a good block cipher, but our plaintext is larger than 128-bit block size

128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

128-bit ciphertext

• What should we do?
Electronic Code Book (ECB) Mode
Gradescope: What properties of ECB aren’t great?
Electronic Code Book (ECB) Mode

- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks
Information Leakage in ECB Mode

Encrypt in ECB mode
Cipher Block Chaining (CBC) Mode: Encryption

- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity
CBC Mode: Decryption

plaintext

Initialization vector

key

decrypt
ciphertext
ECB vs. CBC

AES in ECB mode

AES in CBC mode

Similar plaintext blocks produce similar ciphertext blocks (not good!)
Initialization Vector Dangers

Found in the source code for Diebold voting machines:

\[ \text{DesCBCEncrypt}((\text{des}_c\_\text{block}^*)\text{tmp}, (\text{des}_c\_\text{block}^*)\text{record.m}_\text{Data}, \text{totalSize}, \text{DESKEY}, \text{NULL}, \text{DES}_\text{ENCRYPT}) \]
Counter Mode (CTR): Encryption

- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity; Fragile if ctr repeats
Information Leakage in CTR Mode (poorly)

Encrypt in CTR mode:
But with the same counter for each frame!
Counter Mode (CTR): Decryption

\begin{align*}
\text{Initial } \text{ctr} & \quad \text{ctr} \quad \text{ctr+1} \quad \text{ctr+2} \quad \text{ctr+3} \\
\text{block cipher} & \quad \text{block cipher} \quad \text{block cipher} \quad \text{block cipher} \\
\text{ct} & \quad \oplus & \quad \text{ct} & \quad \oplus & \quad \text{ct} & \quad \oplus & \quad \text{ct} & \quad \oplus \\
\text{pt} & & \text{pt} & & \text{pt} & & \text{pt} & \end{align*}
Ok, so what mode do I use?

• Don’t choose a mode, use established libraries 😊
  • Libsodium’s secretbox encryption solves ‘all the problems’ for example

• Good modes:
  • GCM - Galois/Counter Mode
  • CTR (sometimes)
  • Even ECB is fine in ‘the right circumstance’

• AES-128 is standard
  • Be concerned if something says “AES 1024”...

https://research.kudelskisecurity.com/2022/05/11/practical-bruteforce-of-aes-1024-military-grade-encryption/
When is an Encryption Scheme “Secure”? 

- Hard to recover the key?
  - What if attacker can learn plaintext without learning the key?
- Hard to recover plaintext from ciphertext?
  - What if attacker learns some bits or some function of bits?
How Can a Cipher Be Attacked?

• Attackers knows ciphertext and encryption algorithm
  • What else does the attacker know? Depends on the application in which the cipher is used!

• Ciphertext-only attack
• KPA: Known-plaintext attack (stronger)
  • Knows some plaintext-ciphertext pairs
• CPA: Chosen-plaintext attack (even stronger)
  • Can obtain ciphertext for any plaintext of choice
• CCA: Chosen-ciphertext attack (very strong)
  • Can decrypt any ciphertext except the target
Chosen Plaintext Attack

Crook #1 changes his PIN to a number of his choice

PIN is encrypted and transmitted to bank

cipher(key, PIN)

Crook #2 eavesdrops on the wire and learns ciphertext corresponding to chosen plaintext PIN

... repeat for any PIN value
**Very Informal Intuition**

• Security against chosen-plaintext attack (CPA)
  • Ciphertext leaks no information about the plaintext
  • Even if the attacker correctly guesses the plaintext, he cannot verify his guess
  • Every ciphertext is unique, encrypting same message twice produces completely different ciphertexts
    • Implication: encryption must be randomized or stateful

• Security against chosen-ciphertext attack (CCA)
  • Integrity protection – it is not possible to change the plaintext by modifying the ciphertext
So Far: Achieving Privacy

Encryption schemes: A tool for protecting privacy.

Message = M
Ciphertext = C
Now: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.

Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.
Reminder: CBC Mode Encryption

- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity
CBC-MAC

- Not secure when system may MAC messages of different lengths
- Use a different key – not encryption key
- **NIST recommends a derivative called CMAC [FYI only]**
Another Tool: Hash Functions
Hash Functions: Main Idea

- Hash function $H$ is a lossy compression function
  - Collision: $h(x) = h(x')$ for distinct inputs $x, x'$
- $H(x)$ should look "random"
  - Every bit (almost) equally likely to be 0 or 1
- **Cryptographic hash function** needs a few properties...
Property 1: One-Way

• Intuition: hash should be hard to invert
  • “Preimage resistance”
  • Let $h(x') = y$ in $\{0,1\}^n$ for a random $x'$
  • Given $y$, it should be hard to find any $x$ such that $h(x)=y$

• How hard?
  • Brute-force: try every possible $x$, see if $h(x)=y$
  • SHA-1 (common hash function) has 160-bit output
    • Expect to try $2^{159}$ inputs before finding one that hashes to $y$. 
Property 2: Collision Resistance

• Should be hard to find $x \neq x'$ such that $h(x) = h(x')$
Birthday Paradox

• Are there two people in your part of the classroom that have the same birthday?
  • 365 days in a year (366 some years)
    • Pick one person. To find another person with same birthday would take on the order of $365/2 = 182.5$ people
    • Expect birthday “collision” with a room of only 23 people.
    • For simplicity, approximate when we expect a collision as $\sqrt{365}$.

• Why is this important for cryptography?
  • $2^{128}$ different 128-bit values
    • Pick one value at random. To exhaustively search for this value requires trying on average $2^{127}$ values.
    • Expect “collision” after selecting approximately $2^{64}$ random values.
    • 64 bits of security against collision attacks, not 128 bits.
Property 2: Collision Resistance

- Should be hard to find \(x \neq x'\) such that \(h(x) = h(x')\)
- Birthday paradox means that brute-force collision search is only \(O(2^{n/2})\), not \(O(2^n)\)
  - For SHA-1, this means \(O(2^{80})\) vs. \(O(2^{160})\)
One-Way vs. Collision Resistance

One-wayness does not imply collision resistance.

Collision resistance does not imply one-wayness.

You can prove this by constructing a function that has one property but not the other.
One-Way vs. Collision Resistance (Details here mainly FYI)

• One-wayness does not imply collision resistance
  • Suppose \( g \) is one-way
  • Define \( h(x) \) as \( g(x') \) where \( x' \) is \( x \) except drop the last bit
    • \( h \) is one-way (to invert \( h \), must invert \( g \))
    • Collisions for \( h \) are easy to find: for any \( x \), \( h(x_0) = h(x_1) \)

• Collision resistance does not imply one-wayness
  • Suppose \( g \) is collision-resistant
  • Define \( y = h(x) \) to be \( 0x \) if \( x \) is \( n \)-bit long, \( 1g(x) \) otherwise
    • Collisions for \( h \) are hard to find: if \( y \) starts with 0, then there are no collisions, if \( y \) starts with 1, then must find collisions in \( g \)
    • \( h \) is not one way: half of all \( y \)'s (those whose first bit is 0) are easy to invert (how?); random \( y \) is invertible with probability \( \frac{1}{2} \)
Property 3: Weak Collision Resistance

• Given randomly chosen \( x \), hard to find \( x' \) such that \( h(x)=h(x') \)
  • Attacker must find collision for a specific \( x \). By contrast, to break collision resistance it is enough to find any collision.
  • Brute-force attack requires \( O(2^n) \) time

• Weak collision resistance does not imply collision resistance.
Hashing vs. Encryption

• Hashing is one-way. There is no “un-hashing”
  • A ciphertext can be decrypted with a decryption key... hashes have no equivalent of “decryption”

• Hash(x) looks “random” but can be compared for equality with Hash(x’)
  • Hash the same input twice → same hash value
  • Encrypt the same input twice → different ciphertexts

• Cryptographic hashes are also known as “cryptographic checksums” or “message digests”
Application: Password Hashing

• Instead of user password, store $\text{hash}(\text{password})$
• When user enters a password, compute its hash and compare with the entry in the password file

• Why is hashing better than encryption here?
Application: Password Hashing

• Instead of user password, store $\text{hash}(\text{password})$

• When user enters a password, compute its hash and compare with the entry in the password file

• Why is hashing better than encryption here?

• System does not store actual passwords!

• Don’t need to worry about where to store the key!

• Cannot go from hash to password!
Application: Password Hashing

• Which property do we need?
  • One-wayness?
  • (At least weak) Collision resistance?
  • Both?
Application: Password Hashing + Salting

• **Salting**
  • We ‘salt’ hashes for password by adding a randomized suffix to the password
    • E.g. Hash(“coolpassword”+”35B67C2A”)
  • We then store the salt with the hashed password!
  • Server generates the salt

• The goal is to prevent *precomputation attacks*
  • If the adversary doesn’t know the salt, they can’t *precompute* common passwords
Hash Functions Review

• Map large domain to small range (e.g., range of all 160- or 256-bit values)

• Properties:
  • Collision Resistance: Hard to find two distinct inputs that map to same output
  • One-wayness: Given a point in the range (that was computed as the hash of a random domain element), hard to find a preimage
  • Weak Collision Resistance: Given a point in the domain and its hash in the range, hard to find a new domain element that maps to the same range element
**Application: Software Integrity**

**Goal:** Software manufacturer wants to ensure file is received by users without modification.

**Idea:** given goodFile and hash(goodFile), very hard to find badFile such that hash(goodFile)=hash(badFile)
Application: Software Integrity

- Which property do we need?
  - One-wayness?
  - (At least weak) Collision resistance?
  - Both?
Which Property Do We Need?
One-wayness, Collision Resistance, Weak CR?

• UNIX passwords stored as hash(password)
  • One-wayness: hard to recover the/a valid password

• Integrity of software distribution
  • Weak collision resistance
  • But software images are not really random... may need full collision resistance if considering malicious developers
Which Property Do We Need?

- UNIX passwords stored as hash(password)
  - **One-wayness:** hard to recover the/a valid password
- Integrity of software distribution
  - **Weak collision resistance**
  - But software images are not really random... may need **full collision resistance** if considering malicious developers
- Commitments (e.g. auctions)
  - Alice wants to bid B, sends H(B), later reveals B
  - **One-wayness:** rival bidders should not recover B (this may mean that they need to hash some randomness with B too)
  - **Collision resistance:** Alice should not be able to change their mind to bid B’ such that H(B)=H(B’)

CSE 484 - Spring 2024
Commitments
Common Hash Functions

• SHA-2: SHA-256, SHA-512, SHA-224, SHA-384
• SHA-3: standard released by NIST in August 2015
• MD5 – Don’t Use!
  • 128-bit output
  • Designed by Ron Rivest, used very widely
  • Collision-resistance broken (summer of 2004)
• RIPEMD
  • 160-bit version is OK
  • 128-bit version is not good
• SHA-1 (Secure Hash Algorithm) – Don’t Use!
  • 160-bit output
  • US government (NIST) standard as of 1993-95
  • Theoretically broken 2005; practical attack 2017!
SHA-1 Broken in Practice (2017)

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
Aside: How we evaluate hash functions

• Speed
  • Is it amenable to hardware implementations?

• Diffusion
  • Does changing 1 bit in the input affect all output bits?

• Resistance to attack approaches
  • Collisions?
  • Length extensions?
  • etc
Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.

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? (Historical reasons)
  • 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
MAC with SHA3

• SHA3(Key || Message)

• SHA3 is designed to get the same safety properties as HMAC constructions
Authenticated Encryption

• What if we want **both** privacy and integrity?
• Natural approach: combine encryption scheme and a MAC.

• Is this fine? (Pollev)

\[
\begin{align*}
M_1 & \xrightarrow{\text{Encrypt}_{Ke}} M'_1 \\
M_2 & \xrightarrow{\text{Encrypt}_{Ke}} M'_2 \\
M_3 & \xrightarrow{\text{Encrypt}_{Ke}} M'_3 \\
\end{align*}
\[
\begin{align*}
\text{MAC}_{Km} & \xrightarrow{} T_1 \\
\text{MAC}_{Km} & \xrightarrow{} T_2 \\
\text{MAC}_{Km} & \xrightarrow{} T_3 \\
\end{align*}
\]
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

\[
\begin{align*}
\text{Encrypt}_K & \quad \text{MAC}_K \\
\text{DON'T FIRE} & \quad \text{C'2} & \quad \text{T2} \\
\text{Encrypt}_K & \quad \text{MAC}_K \\
\text{FIRE} & \quad \text{C'3} & \quad \text{T3} \\
\end{align*}
\]
Authenticated Encryption

• Instead: 
  Encrypt *then* MAC.

• (Not as good: MAC-then-Encrypt)