CPA SECURITY and HASH FUNCTIONS
Recall: CPA security

Let $\mathcal{E} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme

<table>
<thead>
<tr>
<th>Game Left$_{\mathcal{E}}$</th>
<th>Game Right$_{\mathcal{E}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure Initialize</td>
<td>procedure Initialize</td>
</tr>
<tr>
<td>$K \leftarrow \mathcal{K}$</td>
<td>$K \leftarrow \mathcal{K}$</td>
</tr>
<tr>
<td>procedure LR($M_0, M_1$)</td>
<td>procedure LR($M_0, M_1$)</td>
</tr>
<tr>
<td>Return $C \leftarrow \mathcal{E}_K(M_0)$</td>
<td>Return $C \leftarrow \mathcal{E}_K(M_1)$</td>
</tr>
</tbody>
</table>

The (ind-cpa) advantage of $A$ is

$$\operatorname{Adv}_{\mathcal{E}}^{\text{ind-cpa}}(A) = \Pr[\text{Right}_{\mathcal{E}}^A \Rightarrow 1] - \Pr[\text{Left}_{\mathcal{E}}^A \Rightarrow 1]$$

**Security:** $\mathcal{E}$ is IND-CPA-secure if $\operatorname{Adv}_{\mathcal{E}}^{\text{ind-cpa}}(A)$ is “small” for ALL $A$ that use “practical” amounts of resources.
Why is IND-CPA the “master” property?

We claim that if encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is IND-CPA secure then the ciphertext hides ALL partial information about the plaintext.

For example, from $C_1 \leftarrow^$ $\mathcal{E}_K(M_1)$ and $C_2 \leftarrow^$ $\mathcal{E}_K(M_2)$ the adversary cannot

- get $M_1$
- get 1st bit of $M_1$
- get XOR of the 1st bits of $M_1, M_2$
- etc.
Birthday attack on CTR

Let \( E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell \) be a family of functions and \( SE = (K, E, D) \) the corresponding CTR symmetric encryption scheme. Suppose 1-block messages \( M_0, M_1 \) are encrypted:

\[
C_0[0] \xleftarrow{\$} E(K, M_0) \quad \text{and} \quad C_1[0] \xleftarrow{\$} E(K, M_1)
\]

Let us say we are **lucky** if \( C_0[0] = C_1[0] \). If so:

\[
C_0[1] = C_1[1] \text{ if and only if } M_0 = M_1
\]

So if we are lucky we can detect message equality and violate IND-CPA.
Let $1 \leq q < 2^n$ be a parameter and let $\langle i \rangle$ be integer $i$ encoded as an $\ell$-bit string.

**adversary $A$**

for $i = 1, \ldots, q$ do

$C^i[0]C^i[1] \leftarrow LR(\langle i \rangle, \langle 0 \rangle)$

$S \leftarrow \{(j, t): C^j[0] = C^t[0] \text{ and } j < t\}$

If $S \neq \emptyset$, then

$(j, t) \leftarrow S$

If $C^j[1] = C^t[1]$ then return 1

return 0
Birthday attack on CTR$: Right game analysis

adversary $A$

for $i = 1, \ldots, q$ do

$C^i[0]C^i[1] \leftarrow \mathsf{LR}(\langle i \rangle, \langle 0 \rangle)$

$S \leftarrow \{(j, t) : C^j[0] = C^t[0] \text{ and } j < t\}$

If $S \neq \emptyset$, then

$(j, t) \leftarrow S$

If $C^j[1] = C^t[1]$ then return 1

return 0

If $C^j[0] = C^t[0]$ (lucky) then

$C^j[1] = \langle 0 \rangle \oplus E_K(C^j[0] + 1) = \langle 0 \rangle \oplus E_K(C^t[0] + 1) = C^t[1]$

SO

$\Pr \left[ \mathsf{Right}_{SE}^A \Rightarrow 1 \right] = \Pr [S \neq \emptyset] = C(2^n, q)$
Birthday attack on CTR$: Left game analysis

adversary $A$

for $i = 1, \ldots, q$ do

$C^i[0]C^i[1] \leftarrow \text{LR}(\langle i \rangle, \langle 0 \rangle)$

$S \leftarrow \{(j, t): C^j[0] = C^t[0] \text{ and } j < t\}$

If $S \neq \emptyset$, then

$(j, t) \leftarrow S$

If $C^j[1] = C^t[1]$ then return 1

return 0

If $C^j[0] = C^t[0]$ (lucky) then

$C^j[1] = \langle j \rangle \oplus E_K(C^j[0] + 1) \neq \langle t \rangle \oplus E_K(C^t[0] + 1) = C^t[1]$

SO

$$\Pr \left[ \text{Left}_{\mathcal{SE}}^A \Rightarrow 1 \right] = 0.$$
Birthday attack on CTR

\[
\text{Adv}^{\text{ind-\text{cpa}}} (A) = \Pr [\text{Right}_A \Rightarrow 1] - \Pr [\text{Left}_A \Rightarrow 1] = C(2^n, q) - 0 \geq 0.3 \cdot \frac{q(q - 1)}{2^n}
\]

**Conclusion:** CTR can be broken (in the IND-CPA sense) in about \(2^{n/2}\) queries, where \(n\) is the block length of the underlying block cipher, regardless of the cryptanalytic strength of the block cipher.
Security of CTR

So far: A $q$-query adversary can break CTR with advantage $\approx \frac{q^2}{2^{n+1}}$

Question: Is there any better attack?
Security of CTR

So far: A $q$-query adversary can break CTR with advantage $\approx \frac{q^2}{2^{n+1}}$

Question: Is there any better attack?

Answer: NO!

We can prove that the best $q$-query attack short of breaking the block cipher has advantage at most

$$\frac{2(q - 1)\sigma}{2^n}$$

where $\sigma$ is the total number of blocks across all messages encrypted.

Example: If $q$ 1-block messages are encrypted then $\sigma = q$ so the adversary advantage is not more than $2q^2/2^n$.

For $E = AES$ this means up to about $2^{64}$ blocks may be securely encrypted, which is good.
Theorem: Let $E : \{0, 1\}^k \times \{0, 1\}^n \to \{0, 1\}^\ell$ be a family of functions and $SE = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ the corresponding CTR symmetric encryption scheme. Let $A$ be an ind-cpa adversary against $SE$ that has running time $t$ and makes at most $q$ LR queries, the messages across them totaling at most $\sigma$ blocks. Then there is a prf-adversary $B$ against $E$ such that

$$\text{Adv}^{\text{ind-cpa}}_{SE}(A) \leq 2 \cdot \text{Adv}^{\text{prf}}_{E}(B) + \frac{2(q - 1)\sigma}{2^n}$$

Furthermore, $B$ makes at most $\sigma$ oracle queries and has running time $t + \Theta(\sigma \cdot (n + \ell))$. 

Nadia Heninger UCSD
We won’t prove this, but let’s give some intuition.

We assume for simplicity that both messages in each LR query of A are $m$ blocks long. Thus $\sigma = mq$.

Note a block is $\ell$ bits, so each message in a query is $ml$ bits.

We let $C_i = C_i[0]C_i[1] \ldots C_i[m]$ denote the response of the LR oracle to A’s $i$-th query.
Consider the CTR scheme with $E_K$ replaced by a random function $F_n$ with range ${0, 1}^\ell$.

```
\textbf{Alg $E_{F_n}(M)$}

\begin{align*}
C[0] & \leftarrow \{0, 1\}^n \\
\text{for } i = 1, \ldots, m \text{ do} & \\
& \quad P[i] \leftarrow F_n(C[0] + i) \\
& \quad C[i] \leftarrow P[i] \oplus M[i] \\
\text{return } C
\end{align*}
```

Analyzing this is a thought experiment, but we can ask whether it is IND-CPA secure.

If so, the assumption that $E$ is a PRF says CTR with $E$ is IND-CPA secure.
CTR$ with a random function

Let $W$ be the event that the points

$$C_1[0] + 1, \ldots, C_1[0] + m, \ldots, C_q[0] + 1, \ldots, C_q[0] + m,$$

on which $F_n$ is evaluated across the $q$ encryptions, are all distinct.

**Case 1:** $W$ happens. Then the encryption is a one-time-pad: ciphertexts are random, independent strings, regardless of which message is encrypted. So $A$ has zero advantage.

**Case 2:** $W$ doesn’t happen. Then $A$ may have high advantage but it does not matter because $\Pr[W]$ doesn’t happen is small. (It is the small additive term in the theorem.)
Theorem: [BDJR97] Let $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher and $SE = (K, E, D)$ the corresponding CBC symmetric encryption scheme. Let $A$ be an ind-cpa adversary against $SE$ that has running time $t$ and makes at most $q$ LR queries, the messages across them totaling at most $\sigma$ blocks. Then there is a prf-adversary $B$ against $E$ such that

$$\text{Adv}_{SE}^{\text{ind-cpa}}(A) \leq 2 \cdot \text{Adv}_E^{\text{prf}}(B) + \frac{\sigma^2}{2^n}$$

Furthermore, $B$ makes at most $\sigma$ oracle queries and has running time $t + \Theta(\sigma \cdot n)$. 
CBC mode is IND-CPA secure, but vulnerable both in theory and practice to chosen ciphertext attacks, which we will cover in future lectures.

Probably best to avoid using it because of the difficulty of implementing it securely.
At death’s door for years, widely used SHA1 function is now dead

Algorithm underpinning Internet security falls to first-known collision attack.

DAN GOODIN - 2/23/2017, 5:01 AM

For more than six years, the SHA1 cryptographic hash function underpinning Internet security has been at death's door. Now it's officially dead, thanks to the submission of the first known instance of a fatal exploit known as a "collision."
New Topic: Hash functions

- MD: MD4, MD5, MD6
- SHA2: SHA1, SHA224, SHA256, SHA384, SHA512
- SHA3: SHA3-224, SHA3-256, SHA3-384, SHA3-512

Their primary purpose is collision-resistant data compression, but they have many other purposes and properties as well ... A hash function is often treated like a magic wand ...

Some uses:
- Certificates: How you know www.snapchat.com really is Snapchat
- Bitcoin
- Data authentication with HMAC: TLS, ...
New Topic: Hash functions

- MD: MD4, MD5, MD6
- SHA2: SHA1, SHA224, SHA256, SHA384, SHA512
- SHA3: SHA3-224, SHA3-256, SHA3-384, SHA3-512

Their primary purpose is collision-resistant data compression, but they have many other purposes and properties as well ... A hash function is often treated like a magic wand ...

Some uses:
- Certificates: How you know www.snapchat.com really is Snapchat
- Bitcoin
- Data authentication with HMAC: TLS, ...

SHA = “Secure Hash Algorithm” 😊
A **collision** for a function $h : D \rightarrow \{0, 1\}^n$ is a pair $x_1, x_2 \in D$ of points such that

- $h(x_1) = h(x_2)$, and
- $x_1 \neq x_2$.

If $|D| > 2^n$ then the pigeonhole principle tells us that there must exist a collision for $h$. 
A **collision** for a function $h : D \rightarrow \{0, 1\}^n$ is a pair $x_1, x_2 \in D$ of points such that

- $h(x_1) = h(x_2)$, and
- $x_1 \neq x_2$.

If $|D| > 2^n$ then the pigeonhole principle tells us that there must exist a collision for $h$. 

![Diagram of collisions](image)
A collision for a function $h : D \to \{0, 1\}^n$ is a pair $x_1, x_2 \in D$ of points such that

- $h(x_1) = h(x_2)$, and
- $x_1 \neq x_2$.

If $|D| > 2^n$ then the pigeonhole principle tells us that there must exist a collision for $h$.

We want that even though collisions exist, they are hard to find.
Collision-resistance of a function family

The formalism considers a family $H : \text{Keys} \times D \rightarrow R$ of functions, meaning for each $K \in \text{Keys}$ we have a function $H_K : D \rightarrow R$ defined by $H_K(x) = H(K, x)$.

<table>
<thead>
<tr>
<th>Game $\text{CR}_H$</th>
<th>procedure Initialize $K \leftarrow$ Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure Finalize($x_1, x_2$)</td>
<td>If ($x_1 = x_2$) then return false</td>
</tr>
<tr>
<td>Return $K$</td>
<td>If ($x_1 \not\in D$ or $x_2 \not\in D$) then return false</td>
</tr>
<tr>
<td>Return ($H_K(x_1) = H_K(x_2)$)</td>
<td></td>
</tr>
</tbody>
</table>

Let

$$\text{Adv}^{\text{cr}}_H(A) = \Pr \left[ \text{CR}_H^A \Rightarrow \text{true} \right].$$
Collision-resistance

Game $\text{CR}_H$

<table>
<thead>
<tr>
<th>procedure Initialize</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K \leftarrow$ Keys</td>
</tr>
<tr>
<td>Return $K$</td>
</tr>
</tbody>
</table>

procedure Finalize($x_1, x_2$)

If ($x_1 = x_2$) then return false
If ($x_1 \notin D$ or $x_2 \notin D$) then return false
Return ($H_K(x_1) = H_K(x_2)$)

The Return statement in Initialize means that the adversary $A$ gets $K$ as input. The key $K$ here is not secret!

Adversary $A$ takes $K$ and tries to output a collision $x_1, x_2$ for $H_K$.

$A$’s output is the input to Finalize, and the game returns true if the collision is valid.
Example

Let \( N = 2^{256} \) and define

\[
H: \{1, \ldots, N\} \times \{0, 1, 2, \ldots\} \rightarrow \{0, 1, \ldots, N - 1\}
\]

by

\[
H(K, x) = (x \mod K).
\]

Q: Is \( H \) collision resistant?
Let $N = 2^{256}$ and define

\[ H: \{1, \ldots, N\} \times \{0, 1, 2, \ldots\} \rightarrow \{0, 1, \ldots, N - 1\} \]

by

\[
H(K, x) = (x \mod K).
\]

**Q:** Is $H$ collision resistant?

**A:** NO!

**Why?** $(x + K) \mod K = x \mod K$

**adversary** $A(K)$

$x_1 \leftarrow 0$; $x_2 \leftarrow K$; Return $x_1, x_2$

\[ \text{Adv}^\text{cr}_H(A) = 1 \]
Let $E: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a blockcipher.
Let $H: \{0,1\}^k \times \{0,1\}^{2n} \rightarrow \{0,1\}^n$ be defined by

**Alg** $H(K, x[1]x[2])$

$y \leftarrow E_K(E_K(x[1]) \oplus x[2])$; Return $y$

Let’s show that $H$ is not collision-resistant by giving an efficient adversary $A$ such that $\text{Adv}^\text{cr}_H(A) = 1$. 
Let $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher.

Let $H: \{0, 1\}^k \times \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ be defined by

\[
\text{Alg } H(K, x[1]x[2]) \\
y \leftarrow E_K(E_K(x[1]) \oplus x[2]); \text{ Return } y
\]

Let’s show that $H$ is not collision-resistant by giving an efficient adversary $A$ such that $\text{Adv}^{cr}_H(A) = 1$.

**Idea:** Pick $x_1 = x_1[1]x_1[2]$ and $x_2 = x_2[1]x_2[2]$ so that

\[
E_K(x_1[1]) \oplus x_1[2] = E_K(x_2[1]) \oplus x_2[2]
\]
Example

Algorithm $H(K, x[1]x[2])$

$y ← E_K(E_K(x[1]) ⊕ x[2])$; Return $y$

Idea: Pick $x_1 = x_1[1]x_1[2]$ and $x_2 = x_2[1]x_2[2]$ so that

$$E_K(x_1[1]) ⊕ x_1[2] = E_K(x_2[1]) ⊕ x_2[2]$$

Adversary $A(K)$

$x_1 ← 0^n1^n; x_2[2] ← 0^n; x_2[1] ← E_K^{-1}(E_K(x_1[1]) ⊕ x_1[2] ⊕ x_2[2])$

return $x_1, x_2$

Then $\text{Adv}^\text{cr}_H(A) = 1$ and $A$ is efficient, so $H$ is not CR.

Note how we used the fact that $A$ knows $K$ and the fact that $E$ is a blockcipher!
Let $E: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ be a blockcipher. Let $D$ be the set of all strings whose length is a positive multiple of $l$.

Define the hash function $H: \{0, 1\}^k \times D \rightarrow \{0, 1\}^l$ as follows:

**Alg** $H(K, M)$


$C[0] \leftarrow 0^l$

For $i = 1, \ldots, n$ do

$B[i] \leftarrow E(K, C[i - 1] \oplus M[i]); C[i] \leftarrow E(K, B[i] \oplus M[i])$

Return $C[n]$

Show that $H$ is not CR by giving an efficient adversary $A$ such that $\text{Adv}^{cr}_H(A) = 1$. 

Nadia Heninger  

UCSD  

30
We say that \( H: \text{Keys} \times D \to R \) is keyless if \( \text{Keys} = \{ \varepsilon \} \) consists of just one key, the empty string.

In this case we write \( H(x) \) in place of \( H(\varepsilon, x) \) or \( H_\varepsilon(x) \).

Practical hash functions like the MD, SHA2 and SHA3 series are keyless.
The hash function SHA256: $\{0, 1\}^{<2^{64}} \rightarrow \{0, 1\}^{256}$ is \textbf{keyless}, with

- Inputs being strings $X$ of any length strictly less than $2^{64}$
- Outputs always having length 256.

\begin{verbatim}
Alg SHA256(X)  // $|X| < 2^{64}$
M ← shapad(X)  // $|M| \mod 512 = 0$
$M^{(1)}M^{(2)}\ldots M^{(n)} ← M$  // Break $M$ into 512 bit blocks
$H^{(0)}_0 ← 6a09e667; H^{(0)}_1 ← bb67ae85; \ldots; H^{(0)}_7 ← 5be0cd19$
$H^{(0)} ← H^{(0)}_0H^{(0)}_1\ldots H^{(0)}_7$  // $|H^{(0)}_i| = 32, |H^{(0)}| = 256$
For $i = 1, \ldots, n$ do $H^{(i)} ← \text{sha256}(M^{(i)} \parallel H^{(i-1)})$
Return $H^{(n)}$
\end{verbatim}

sha256: $\{0, 1\}^{512+256} \rightarrow \{0, 1\}^{256}$ is the \textbf{compression function}. 
Padding, and initialization vector $H^{(0)}$

**Alg** `shapad(X)`  // $|X| < 2^{64}$

$d \leftarrow (447 - |X|) \mod 512$  // Chosen to make $|M|$ a multiple of 512

Let $\ell$ be the 64-bit binary representation of $|X|$

$M \leftarrow X \parallel 1 \parallel 0^d \parallel \ell$  // $|M|$ is a multiple of 512

return $M$

The 32-bit word $H_j^{(0)}$ was obtained by taking the first 32 bits of the fractional part of the square root of the $j$-th prime number ($0 \leq j \leq 7$).
Padding, and initialization vector $H^{(0)}$

**Algorithm** \( \text{shapad}(X) \)  
\[ d \leftarrow (447 - |X|) \mod 512 \]  
\[ \text{Chosen to make } |M| \text{ a multiple of 512} \]

Let \( \ell \) be the 64-bit binary representation of \( |X| \)

\[ M \leftarrow X \parallel 1 \parallel 0^d \parallel \ell \]  
\[ \text{ // } |M| \text{ is a multiple of 512} \]

return \( M \)

The 32-bit word \( H_j^{(0)} \) was obtained by taking the first 32 bits of the fractional part of the square root of the \( j \)-th prime number \( (0 \leq j \leq 7) \).

**Question:** Why square roots as opposed to simply picking the words at random and embedding them in the code?

**Speculation:** Perhaps to prevent suspicion of subversion (planting of a backdoor)?
Compression function sha256

Compression function sha256: \( \{0, 1\}^{512+256} \to \{0, 1\}^{256} \) takes a 512 + 256 = 768 bit input and returns a 256-bit output.

\[
\textbf{Alg} \quad \text{sha256}(x\|v) \quad // \quad |x|=512, \quad |v|=256
\]

\[
w \leftarrow \text{E}^{\text{sha256}}(x, v)
\]

\[
w_0 \cdots w_7 \leftarrow w \quad // \quad \text{Break } w \text{ into 32-bit words}
\]

\[
v_0 \ldots v_7 \leftarrow v \quad // \quad \text{Break } v \text{ into 32-bit words}
\]

For \( j = 0, \ldots, 7 \) do \( h_j \leftarrow w_j + v_j \)

\[
h \leftarrow h_0 \ldots h_7 \quad // \quad |h| = 256
\]

Return \( h \)

Here and on next slide, “+” denotes addition modulo \( 2^{32} \).

\( \text{E}^{\text{sha256}}: \quad \{0, 1\}^{512} \times \{0, 1\}^{256} \to \{0, 1\}^{256} \) is a block cipher with 512-bit keys and 256-bit blocks.
Block cipher $E_{sha256}$

\[
\textbf{Alg } E_{sha256}(x, v) \quad // \quad x \text{ is a 512-bit key, } v \text{ is a 256-bit input}
\]

\[
x_0 \cdots x_{15} \leftarrow x \quad // \quad \text{Break } x \text{ into 32-bit words}
\]

\[
\text{For } t = 0, \ldots, 15 \text{ do } W_t \leftarrow x_t
\]

\[
\text{For } t = 16, \ldots, 63 \text{ do } W_t \leftarrow \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}
\]

\[
v_0 \cdots v_7 \leftarrow v \quad // \quad \text{Break } v \text{ into 32-bit words}
\]

\[
\text{For } j = 0, \ldots, 7 \text{ do } S_j \leftarrow v_j \quad // \quad \text{Initialize 256-bit state } S
\]

\[
\text{For } t = 0, \ldots, 63 \text{ do} \quad // \quad 64 \text{ rounds}
\]

\[
T_1 \leftarrow S_7 + \gamma_1(S_4) + Ch(S_4, S_5, S_6) + C_t + W_t
\]

\[
T_2 \leftarrow \gamma_0(S_0) + Maj(S_0, S_1, S_2)
\]

\[
S_7 \leftarrow S_6 ; S_6 \leftarrow S_5 ; S_5 \leftarrow S_4 ; S_4 \leftarrow S_3 + T_1
\]

\[
S_3 \leftarrow S_2 ; S_2 \leftarrow S_1 ; S_1 \leftarrow S_0 ; S_0 \leftarrow T_1 + T_2
\]

\[
S \leftarrow S_0 \cdots S_7
\]

\[
\text{Return } S \quad // \quad 256\text{-bit output}
\]
Internals of block cipher $E_{sha256}$

On the previous slide:

- $\sigma_0, \sigma_1, \gamma_0, \gamma_1, Ch, Maj$ are functions not detailed here.
- $C_1 = 428a2f98, C_2 = 71374491, \ldots, C_{63} = c67178f2$ are constants, where $C_i$ is the first 32 bits of the fractional part of the cube root of the $i$-th prime.
SHA256 hash calculator

http://www.xorbin.com/tools/sha256-hash-calculator

**SHA-256** produces a 256-bit (32-byte) hash value.

**Data**

CSE 107 is way too easy!

**SHA-256 hash**

7263ee434ed956bb9c70b580465f49923eb2c39677a9c862d536a42798db96f

Calculate SHA256 hash
Usage of hash functions

Uses include hashing the data before signing in creation of certificates, data authentication with HMAC, key-derivation, Bitcoin, ...

These will have to wait, so we illustrate another use, the hashing of passwords.
Authentication via passwords

- Client $A$ has a password $PW$ that is also stored by server $B$
- $A$ authenticates itself by sending $PW$ to $B$ over a secure channel (TLS)

$$A^{PW} \xrightarrow{PW} B^{PW}$$

**Problem:** The password will be found by an attacker who compromises the server.

These types of server compromises are common and often in the news: Yahoo, Equifax, ...
Hashed passwords

- Client $A$ has a password $PW$ and server stores $\overline{PW} = H(PW)$.
- $A$ sends $PW$ to $B$ (over a secure channel) and $B$ checks that $H(PW) = \overline{PW}$

\[ A^{PW} \rightarrow PW \rightarrow B^{PW} \]

Server compromise results in attacker getting $\overline{PW}$ which should not reveal $PW$ as long as $H$ is one-way, which is a consequence of collision-resistance.

But we will revisit this when we consider dictionary attacks!

This is how client authentication is done on the Internet, for example login to gmail.com.
Birthday collision-finding attack

Let $H : \{0, 1\}^k \times D \rightarrow \{0, 1\}^n$ be a family of functions with $|D| > 2^n$. The $q$-trial birthday attack is the following adversary $A_q$ for game $CR_H$:

**adversary** $A_q(K)$

for $i = 1, \ldots, q$ do $x_i \leftarrow D$ ; $y_i \leftarrow H_K(x_i)$

if $\exists i, j$ ($i \neq j$ and $y_i = y_j$ and $x_i \neq x_j$) then return $x_i, x_j$

else return ⊥

Interestingly, the analysis of this via the birthday problem is not trivial, but it shows that

$$\text{Adv}_{CR}^H(A_q) \geq 0.3 \cdot \frac{q(q - 1)}{2^n}.$$  

So a collision can usually be found in about $q = \sqrt{2^n}$ trials.
Birthday attack times

<table>
<thead>
<tr>
<th>Function</th>
<th>$n$</th>
<th>$T_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD4</td>
<td>128</td>
<td>$2^{64}$</td>
</tr>
<tr>
<td>MD5</td>
<td>128</td>
<td>$2^{64}$</td>
</tr>
<tr>
<td>SHA1</td>
<td>160</td>
<td>$2^{80}$</td>
</tr>
<tr>
<td>SHA256</td>
<td>256</td>
<td>$2^{128}$</td>
</tr>
<tr>
<td>SHA512</td>
<td>512</td>
<td>$2^{256}$</td>
</tr>
<tr>
<td>SHA3-256</td>
<td>256</td>
<td>$2^{128}$</td>
</tr>
<tr>
<td>SHA3-512</td>
<td>512</td>
<td>$2^{256}$</td>
</tr>
</tbody>
</table>

$T_B$ is the number of trials to find collisions via a birthday attack.

Design of hash functions aims to make the birthday attack the best collision-finding attack, meaning it is desired that there be no attack succeeding in time much less than $T_B$. 
A compression function is a family $h : \{0, 1\}^k \times \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n$ of functions whose inputs are of a fixed size $b + n$, where $b$ is called the block size.

E.g. $b = 512$ and $n = 256$, in which case

$$h : \{0, 1\}^k \times \{0, 1\}^{768} \rightarrow \{0, 1\}^{256}$$
The MD transform

Let \( h : \{0, 1\}^k \times \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n \) be a compression function with block length \( b \). Let \( D \) be the set of all strings of at most \( 2^b - 1 \) blocks.

The **MD transform** builds from \( h \) a family of functions

\[
H : \{0, 1\}^k \times D \rightarrow \{0, 1\}^n
\]

such that: **If \( h \) is CR, then so is \( H \).**

The problem of hashing long inputs has been reduced to the problem of hashing fixed-length inputs.

There is no need to try to attack \( H \). You won’t find a weakness in it unless \( h \) has one. That is, \( H \) is **guaranteed** to be secure **assuming** \( h \) is secure.

For this reason, MD is the design used in many hash functions, including the MD and SHA2 series. SHA3 uses a different paradigm.
MD setup

Given: Compression function \( h : \{0,1\}^k \times \{0,1\}^{b+n} \to \{0,1\}^n \).

Build: Hash function \( H : \{0,1\}^k \times D \to \{0,1\}^n \).

Since \( M \in D \), its length \( \ell = |M| \) is a multiple of the block length \( b \). We let \( \|M\|_b = |M|/b \) be the number of \( b \)-bit blocks in \( M \), and parse as

\[
M[1] \ldots M[\ell] \leftarrow M.
\]

Let \( \langle \ell \rangle \) denote the \( b \)-bit binary representation of \( \ell \in \{0, \ldots, 2^b - 1\} \).
Given: Compression function $h : \{0, 1\}^k \times \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n$. 

Build: Hash function $H : \{0, 1\}^k \times D \rightarrow \{0, 1\}^n$.

**Alg $H(K, M)$**

$m \leftarrow \|M\|_b$ ; $M[m + 1] \leftarrow \langle m \rangle$ ; $V[0] \leftarrow 0^n$

For $i = 1, \ldots, m + 1$ do

$V[i] \leftarrow h_K(M[i]||V[i - 1])$

Return $V[m + 1]$
**Theorem:** Let \( h : \{0, 1\}^k \times \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n \) be a family of functions and let \( H : \{0, 1\}^k \times D \rightarrow \{0, 1\}^n \) be obtained from \( h \) via the MD transform. Given a cr-adversary \( A_H \) we can build a cr-adversary \( A_h \) such that

\[
\text{Adv}^\text{cr}_H(A_H) \leq \text{Adv}^\text{cr}_h(A_h)
\]

and the running time of \( A_h \) is that of \( A_H \) plus the time for computing \( H \) on the outputs of \( A_H \).

**Implication:**

\( h \text{ CR} \Rightarrow \text{Adv}^\text{cr}_h(A_h) \text{ small} \)

\( \Rightarrow \text{Adv}^\text{cr}_H(A_H) \text{ small} \)

\( \Rightarrow H \text{ CR} \)
A candidate compression function

Let \( E : \{0, 1\}^b \times \{0, 1\}^n \rightarrow \{0, 1\}^n \) be a block cipher. Let us define keyless compression function \( h : \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n \) by

\[
h(x\|v) = E_x(v) .
\]

**Question:** Is \( h \) collision resistant?
A candidate compression function

Let $E : \{0, 1\}^b \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher. Let us define keyless compression function $h : \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n$ by

$$h(x\|v) = E_x(v) .$$

**Question:** Is $h$ collision resistant?

We seek an adversary that outputs distinct $x_1\|v_1,$ $x_2\|v_2$ satisfying

$$E_{x_1}(v_1) = E_{x_2}(v_2) .$$
A candidate compression function

Let $E : \{0, 1\}^b \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher. Let us define keyless compression function $h : \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n$ by

$$h(x \| v) = E_x(v) .$$

**Question:** Is $h$ collision resistant?

We seek an adversary that outputs distinct $x_1 \| v_1, x_2 \| v_2$ satisfying

$$E_{x_1}(v_1) = E_{x_2}(v_2) .$$

**Answer:** NO, $h$ is NOT collision-resistant, because the following adversary $A$ has $\text{Adv}_{h}^{\text{cr}}(A) = 1$:

**adversary $A$**

$x_1 \leftarrow 0^b ; x_2 \leftarrow 1^b ; v_1 \leftarrow 0^n ; y \leftarrow E_{x_1}(v_1) ; v_2 \leftarrow E_{x_2}^{-1}(y)$

Return $x_1 \| v_1 , x_2 \| v_2$
Let $E : \{0, 1\}^b \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher. Let us define keyless compression function $h : \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n$ by

$$h(x \| v) = E_x(v) \oplus v.$$ 

**Question:** Is $h$ collision resistant?
The Davies-Meyer compression function

Let $E : \{0, 1\}^b \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher. Let us define keyless compression function $h : \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n$ by

$$h(x \parallel v) = E_x(v) \oplus v.$$  

**Question:** Is $h$ collision resistant?

We seek an adversary that outputs distinct $x_1 \parallel v_1$, $x_2 \parallel v_2$ satisfying

$$E_{x_1}(v_1) \oplus v_1 = E_{x_2}(v_2) \oplus v_2.$$  

**Answer:** Unclear how to solve this equation, even though we can pick all four variables.
Let $E : \{0, 1\}^b \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher. Let us define keyless compression function $h : \{0, 1\}^{b+n} \rightarrow \{0, 1\}^n$ by

$$h(x \parallel v) = E_x(v) \oplus v.$$

This is called the Davies-Meyer method and is used in the MD and SHA2 series of hash functions, modulo that the $\oplus$ may be replaced by addition.

In particular the compression function sha256 of SHA256 is underlain in this way by the block cipher $E^{sha256} : \{0, 1\}^{512} \times \{0, 1\}^{256} \rightarrow \{0, 1\}^{256}$ that we saw earlier, with the $\oplus$ being replaced by component-wise addition modulo $2^{32}$. 

Nadia Heninger

UCSD
So far we have looked at attacks that do not attempt to exploit the structure of $h$.

Can we get better attacks if we do exploit the structure?

Ideally not, but hash functions have fallen short!
Cryptanalytic attacks against hash functions

<table>
<thead>
<tr>
<th>When</th>
<th>Against</th>
<th>Time</th>
<th>Who</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993, 1996</td>
<td>md5</td>
<td>$2^{16}$</td>
<td>[dBBo, Do]</td>
</tr>
<tr>
<td>2004</td>
<td>MD5</td>
<td>1 hour</td>
<td>[WaFeLaYu]</td>
</tr>
<tr>
<td>2005, 2006</td>
<td>MD5</td>
<td>1 minute</td>
<td>[LeWadW, Ki]</td>
</tr>
<tr>
<td>2005</td>
<td>SHA1</td>
<td>$2^{69}$</td>
<td>[WaYiYu]</td>
</tr>
<tr>
<td>2017</td>
<td>SHA1</td>
<td>$2^{63.1}$</td>
<td>[SBKAM]</td>
</tr>
</tbody>
</table>

Collisions found in compression function md5 of MD5 did not yield collisions for MD5, but collisions for MD5 are now easy.

2017: Google, Microsoft and Mozilla browsers stop accepting SHA1-based certificates.

The SHA256 and SHA512 hash functions are still viewed as secure, meaning the best known attack is the birthday attack.
Crypto breakthrough shows Flame was designed by world-class scientists

The spy malware achieved an attack unlike any cryptographers have seen before.

DAN GOODIN - 6/7/2012, 11:20 AM

The Flame espionage malware that infected computers in Iran achieved mathematical breakthroughs that could only have been accomplished by world-class cryptographers, two of the world's foremost cryptography experts said.

"We have confirmed that Flame uses a yet unknown MD5 chosen-prefix collision attack," Marc Stevens wrote in an e-mail posted to a cryptography discussion group earlier this week. "The collision attack itself is very interesting from a scientific viewpoint, and there are already some practical implications." Benne de Weger, a Stevens colleague and another expert in cryptographic collision

**Flame**
- Revealed: Stuxnet “beta’s” devious alternate attack on Iran nuke program
- Massive espionage malware targeting governments undetected for 5 years
- Iranian computers targeted by new malicious data wiper program
- New in-the-wild malware

Enlarge / An overview of a chosen-prefix collision. A similar technique was used by the Flame espionage malware that targeted Iran. The scientific novelty of the malware underscored the sophistication of malware sponsored by wealthy nation states.
SHA1 collision: https://shattered.io/

Here are two PDF files that display different content, yet have the same SHA-1 digest.
A SHA1 certificate

auth.resnet.ucsb.edu

Issued by: InCommon Server CA
Expired: Friday, June 23, 2017 at 4:59:59 PM Pacific Daylight Time

This certificate is marked as trusted for this account

Trust

Details

Subject Name
Country or Region US
Postal Code 93106
State/Province CA
Locality Santa Barbara
Organization University of California, Santa Barbara
Organizational Unit Housing & Residential Services
Common Name auth.resnet.ucsb.edu

Issuer Name
Country or Region US
Organization Internet2
Organizational Unit InCommon
Common Name InCommon Server CA

Serial Number 36 9D 1F DF 64 38 1F BC 31 0D 00 AA EB 0E 41 50
Version 3
Signature Algorithm SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)
Parameters None

Not Valid Before Monday, June 23, 2014 at 5:00:00 PM Pacific Daylight Time
Not Valid After Friday, June 23, 2017 at 4:59:59 PM Pacific Daylight Time

Public Key Info
Algorithm RSA Encryption (1.2.840.113549.1.1.1)
Parameters None
Public Key 256 bytes: F7 06 04 61 BF 17 3C 4F ...
Exponent 65537
Key Size 2,048 bits
Key Usage Encrypt, Verify, Wrap, Derive
Signature 256 bytes: 19 13 34 94 90 82 D7 E2 ...
Microsoft will cease support for TLS certs signed by SHA1

May 5, 2016

Microsoft announced it will soon cease support for TLS certificates signed by the SHA1 hashing algorithm, according to ArsTechnica.

After hinting in November that it might, the tech giant made it official last week. The end was expected following new research that revealed the popular cryptographic algorithm was susceptible to collision attacks – in which miscreants attempt to find two inputs producing the same hash value. Should they succeed, they would be able to forge digital signatures.
Implications for Bitcoin?

Who Broke the SHA1 Algorithm (And What Does It Mean for Bitcoin)?

The cryptography world has been buzzing with the news that researchers at Google and CWI Amsterdam have succeeded in successfully generating a ‘hash collision’ for two different documents using the SHA1 encryption algorithm, rendering the algorithm ‘broken’ according to cryptographic standards.

But what does this mean in plain language, and what are the implications for the bitcoin network?

Hash collisions

As laid out in a recent CoinDesk explainer, a hash function (of which SHA1 is an example) is used to take a piece of data of any length, process it, and return another piece of data – the ‘hash digest’ – with a fixed length.

One way that hash functions are used in computing is to check whether the contents of files are identical: as long as a hash function is secure, then two files which hash to the same value will always have the same contents.

However, a hash collision occurs when two different files hash to the same value.

Given the mathematical laws that govern hash functions, it is inevitable that hash collisions will occur for some values of input data (because the range of data you could put into the hash function is potentially infinite, but the output length is fixed).

For a secure hash function, the probability of this should be so small that, in practice, it is not possible to make a sufficient number of calculations to find it.

The significance of the Google/CWI team’s results is in the fact that they were able to create a hash collision by finding a much more efficient method – 100,000 times more efficient in fact – than simply guessing every possible value of data.
Why don’t cryptographers build secure hash functions?
Why don’t cryptographers build secure hash functions?

Assess their job performance in light of attacks by selecting a grade below:

A – Cryptographers are doing super well
B – They are OK
C – They suck
F – Just fire them all and give the job to AI
Why don’t cryptographers build secure hash functions?
Why don’t cryptographers build secure hash functions?

Cryptographers seem **perfectly capable** of building secure hash functions. The difficulty is that they strive for **VERY HIGH SPEED**.

SHA256 can run at 3.5 cycles/byte (eBACS: 2018 Intel Core i3-8121U, https://bench.cr.yp.to/results-hash.html) or 0.6 ns per byte, and hardware will make it even faster.

It is **AMAZING** that one gets ANY security at such low cost.

If you allow cryptographers a 10x slowdown, they can up rounds by 10x and designs seem almost impossible to break.
National Institute for Standards and Technology (NIST) held a world-wide competition to develop a new hash function standard.

Contest webpage:

Requested parameters:

- Design: Family of functions with 224, 256, 384, 512 bit output sizes
- Security: CR, one-wayness, near-collision resistance, others...
- Efficiency: as fast or faster than SHA2-256
SHA3

Submissions: 64

Round 1: 51


SHA3: 1: Keccak
SHA3: The Sponge construction

$f: \{0, 1\}^{r+c} \rightarrow \{0, 1\}^{r+c}$ is a (public, invertible!) permutation.

$d$ is the number of output bits, and $c = 2d$.

SHA3 does not use the MD paradigm used by the MD and SHA2 series.

Shake$(M, d)$— Extendable-output function, returning any given number $d$ of bits.