Introduction to Cryptography
Subject 2: Hashing

Prof. Amir Herzberg
Computer Science Department, Bar Ilan University
http://amir.herzberg.name

© Amir Herzberg, 2003-4. Permission is granted for academic use without modification. For other use please contact author.
Outline

- Crypto-Hash properties
- Using and Collecting Randomness
- Random Oracle Method
- Hashing and Randomness
- Confidentiality properties
- One-way functions
- Login using Hashing
- Integrity & Collision Resistance Properties

- Collision Resistant Hash Functions (CRHF)
- Design of CRHF
- Merkle-Damgard construction
- Standard hash functions
- Conclusions
Crypto-Hash Functions - `Wish List`

- **Compression**
  - Unbounded/Long input
  - Short (finite) output

- **Confidentiality**
  - Can’t find $x$ from $h(x)$

- **Collision-resistance**
  - `Strong`: can’t find $x, x'$ s.t. $h(x) = h(x')$
  - `Weak`: given $x$, can’t find $x' \neq x$ s.t. $h(x) = h(x')$

- **Randomness**: uniform output distribution
Using and Collecting Randomness

- Randomness required for cryptography:
  - Keys (esp. consider `one time pad`!)
  - Randomized cryptographic functions (e.g. encryption)
  - Random challenge (to verify freshness)
  - Seed to Pseudo-Random Generator
    - Or key to Pseudo-Random Permutation / Cipher / Function

- Surprisingly hard to collect!
  - Physical devices/chips (e.g., sample radio): expensive, slow, unavailable to software
  - Measuring human actions: slow and requires human interaction
  - Disk/computer status: predictable/observable?
  - All: bits often biased and dependent
Extracting Randomness

- Extraction of uniformly-distributed bits from `weak random bits` was studied extensively
- Requiring input with minimal entropy not enough...
  - Can’t extract randomness by deterministic process!
  - Can extract random bits from entropy if we have random seed
    - Not relevant for applied crypto; if we have seed we can use PRG
- Without random seed… need stronger assumption on input
  - Extensive research on different assumptions…
Using Biased but Independent Bits

- If bits only biased but independent...
  - Sample bit-pairs $x \in \{00, 01, 10, 11\}$
  - Output $\{0$ if $x=01$, $1$ if $x=10$, nothing if $x=00$ or $x=11\}$
  - Why this is Ok?
  - Why not e.g. $z = x[0] \oplus x[1]$?
  - Why inputs have to be independent?

- But: too few inputs... dependency among inputs...

- Much research on extraction using weaker assumptions

- While in practice...
Collecting Randomness in Practice

- Use available sources with some randomness
  - Timing of `unpredictable, unobservable` events
- Extract random seed \((n\) bits)
  - In practice: usually using `cryptographic hash function`
- Use PRG to generate sufficient random bits
- What’s a crypto hash function?
- A random function seems ideal...
  - If we could simply pick one…
Random Oracle Methodology

- Consider any crypto protocol / mechanism, with a cryptographic hash function $h()$
- Analyze as if $h()$ is a random function
  - Of course an invalid assumption as $h()$ is fixed!
  - Whenever $h()$ is used, we call oracle for the random function (black box containing random function)
- Good for screening insecure solutions
- Security under random oracle implies security to many (not all!!) attacks
- Not a complete proof of security, but useful argument/evidence of security.
- Can’t be a criteria for a candidate crypto hash…
Outline

- Crypto-Hash properties
- Using and Collecting Randomness
- Random Oracle Method
- Hashing and Randomness
- Confidentiality properties
- One-way functions
- Login using Hashing
- Integrity & Collision Resistance Properties
- Collision Resistant Hash Functions (CRHF)
- Design of CRHF
- Merkle-Damgard construction
- Standard hash functions
- Conclusions
What’s a hash function?

- An efficient function \( h() \) with:
  - Infinite domain \( \{0,1\}^* \)
  - Finite range \( \{1…m\} / \{0,1\}^n \)

- Main goal: compression
  - Fixed input length

\[ h : \{0,1\}^n \rightarrow \{0,1\}^n \]

- Used extensively, e.g. in compilation, DB

- Goal: Efficient access to data (using sparse indices)…
Efficient Access using Sparse Index

- Huge namespace (sparse index)
  - Examples: names, words…

- Very small fraction actually used
  - E.g., `ZZRUTTX` is not a word

- Idea: fast mapping of words to bins

- Alphabetic bins (first letters)?
  - Not well distributed
  - Some bins with many collisions, e.g. AAA, COH…
  - Some bins empty (e.g. ZZR…)

- Better: use random mapping
  - Each bin will get same # of words (on average)
Hashing for Efficient Access

- Put word $w$ in bin $h(w)$,
  - If $h$ is random, each bin get same number of words
  - We can’t use a truly random $h$… Selecting, storing and using it is infeasible…

- Idea: use a hash function $h$
  - Basic goal: spread words evenly among bins
  - `Random` mapping
  - Avoid / minimize collisions: $h(w)=h(w')$
  - Extracting randomness from the input

- But once we picked $h$, it is not random…
  - If all input is from $h^{-1}(2)$, it is all mapped to 2…
Simple compression (FIL hash) functions

- 4 byte (32 bit) inputs, 2 byte (16 bit) output
- \( h(x) = x^2 \mod 2^{16} \)
- \( h(x) = \text{int}(32768 \times \text{fraction}(x \times a)), \) where \( 0 < a < 1 \)
- \( h(x) = \text{int}(\sqrt{x}) \)
- \( h(x) = x \times 32567 \mod 32767 \)
- \( h(x) = x[0...15] \oplus x[16...31] \)

(Almost) Regular functions: For any \( y, y' \), the number of pre-images of \( y \) and \( y' \) is (almost) equal

- If input is uniformly-distributed, output is uniformly-distributed
- The choice of hash depends on input distribution
For known input distribution $P(x)$, select `optimal` $h$ to fill bins uniformly.

For `good` input distribution, independent of $h$, use `standard` hashes, e.g. regular function.

Problem: for every $h$ there are `bad` input distributions, e.g. $h^{-1}(5)$...

What if adversary (partially or fully) controls input distribution?

- In crypto/security, adversary is malicious!
Randomness from Adversarial Inputs

- **Keyed Hash:**
  - use public key $k$: $h_k(x)$
  - Adversary, input distribution fixed *independently of k*

- **Cryptanalysis-resistant Hash $h$:**
  - Cryptanalysis found no `natural, feasible` yet `bad` input distributions (creating non-uniform output)
  - Cryptanalysis found no collision $h(x)=h(y)$
  - Collisions and `bad` input distributions *exist…*
    - E.g. input: $x$ s.t. the most significant bit of $h(x)$ is zero
Cryptanalysis Resistant Hash

- A fixed, standard hash function $h$ for which cryptanalysis efforts failed to find `natural, feasible` yet `bad` input distributions or collisions

- Random oracle model: analyze as if using random function
  - Justify: no `natural but bad` distribution found (so far?)
  - Useful but clearly a simplification (not a proof)

- Partially-controlled input model: assume that part of the input is not controlled by adversary

- Collision-resistant hash??
Uniform Output vs. Collision-resistance?

- Many collisions will cause non-uniform output
  - But uniform output is possible with several collision
- Collision-resistance is not *sufficient* for uniform output
- Example:
  - Assume we found no collisions in $h(x) = h(y)$ with $x \neq y$
  - Define $h'(x) = 0 || h(x)$
  - Clearly $h'$ also has no collisions
  - But $h'$ has very non-uniform output (MSb of $h'(x)$ is zero!)
- Test practical hash functions also for...
  - Partial collisions and
  - Non-uniform output (for `natural but bad’ input distribution)
- Open question: define *keyless* uniform-output hash
Keyed Hash Functions $h_k(x)$

- The key $k$ is public
  - Like a key (index) of DB, not crypto key
- Two types:
  - Keyed Collision Resistant Hash Functions:
    - Adversary algorithm is fixed before/independently of $k$
    - Adversary cannot efficiently find any collisions $h_k(x)=h_k(y)$
    - Collision-resistance not sufficient for uniform output
  - Universal Hash Functions (UHF):
    - Adversary selects input distribution before/independently of the choice of $k$
Universal Hash Functions (UHF)

- Adversary selects input distribution before/independently of the choice of $k$
- For random $k$ and random $x, y$, the probability of $h_k(x) = h_k(y)$ is $1/m$ $[\pm \text{negl}(k)$ for almost universal$]$
  - $\text{negl}(k): \{ f \mid \text{all polynomial } P>0 ) \text{ exist } K ) \text{ all } k>K f(k)<P(k)\}$
- Output is uniformly distributed (or almost uniformly distributed, for `almost UHF``)
- Simple, efficient functions

**Example**: break input to $r$ words, $x=<x_0,\ldots,x_r>$ where $x_i<m$. Select random $a=<a_0,\ldots,a_r>$ where $a_i<m$. Then $h_a(x) = \sum_{i=0} a_i x_i \mod m$ is a UHF.
Outline

- Crypto-Hash properties
- Using and Collecting Randomness
- Random Oracle Method
- Hashing and Randomness
- Confidentiality properties
- One-way functions
- Login using Hashing
- Integrity & Collision Resistance Properties

- Collision Resistant Hash Functions (CRHF)
- Design of CRHF
- Merkle-Damgard construction
- Standard hash functions
- Conclusions
Confidentiality of Hash

- Hash has no secret key
  - Cannot use to send secret message
- But hash should hide input (pre-image)
  - Cannot learn input given output
  - One Way Function (OWF)
  - A very basic crypto function – much research on what can be constructed from OWF
OWF: hiding the pre-image

- Intuitive requirements:
  - Easy to compute (efficiently), yet
  - Can’t find $x$ from $f(x)$

- Problems:
  - What if $f(x) = \text{constant}$?
    - Can’t find $x$, but clearly not OWF…
    - Redefine goal: Infeasible to find $x’$ s.t. $f(x’) = f(x)$
  - How is $x$ chosen?
    - Uniformly… from $\{0,1\}^*$? (infinite set!)
    - Can we protect specific (not random) choices of $x$?
  - Can we protect partial information about $x$?
Definition: One Way Function (OWF)

- Definition: \( f() \) is a **OWF** if:
  - It is easy to compute (efficiently), and
  - For every length \( l \) (sufficiently long – \( l > l_{min} \)), given \( f(x) \) for \( x \in_R \{0,1\}^l \), it is infeasible to find \( x' \) s.t. \( f(x') = f(x) \).
  - If \( f: \{0,1\}^* \rightarrow \{0,1\}^n \) then \( f \) is a **One Way Hash Function (OWHF)**
More precise def of OWF...

- \( f \) is OWF (One Way Function) if:
  - \( f \) is computed by some Prob. Poly Time (PPT) algorithm,
  - \( \text{ADV}^{\text{OWF}}_{A,f}(n) = \text{Prob}[f(A(f(x))) = f(x) : x \in \mathbb{R}\{0,1\}^n] \)
  - Concrete security: \( \text{ADV}^{\text{OWF}}_{f,n}(t) = \max_A \{ \text{runtime}(A) < t : \text{ADV}^{\text{OWF}}_{A,f}(n) \} \)
  - Asymptotic security: for any PPT alg. \( A \): \( \text{ADV}^{\text{OWF}}_{A,f}(n) \in \text{negl} \)

\[
\text{negl} = \left\{ f : \mathbb{N} \to \mathbb{R} \mid \left( \forall \text{poly} P(\bullet) > 0 \right) \left( \exists \tilde{n} \right) \left( \forall n > \tilde{n} \right) |f(n)| < P(n) \right\}
\]

- Remaining problems:
  - Can we protect specific (not random) choices of \( x \)?
  - Can we protect partial information about \( x \) (e.g. LSb)?
- Trivial solution: use `random oracle analysis`
Perfectly One-Way Prob. Hashing

- **Goal:** definition for `hashing that hides all information about the pre-image`?
  - Not random oracle…

- **Canetti [C97] defined Oracle hashing**
  - Later renamed Perfectly One-Way Probabilistic Hashing
  - **A randomized algorithm, not a function**
  - Reveals `nothing new` about the pre-image
  - Pre-image distribution `well-spread` (e.g. uniform)

- **We will not get into the details**
Applications of OWHF: Unix Password Identification

- Unix Login Mechanism
- Unix passwords file is public (or `less secret`)
- Keep only hash of passwords
- Concern: most users use bad passwords; attacker can hash all of them (dictionary attack)
- Solution: use different hash functions
  - $salt$ is a 12 bit random value
  - Password file contains $<salt, h_{salt}(password)>$
- Unix hash function: 25 applications of DES over block of 64 zero bits, with the password as key
  - With $salt$ modifying one of the DES tables
Unix Passwords - Criticism

1. Hiding of passwords
   - Passwords are non-uniform
   - Exposing partial info may be critical
   - Requires more than OWHF… (`random oracle` model?)

2. Modified, non-standard DES and construction are used – is it secure? Few (positive) results recently.

3. Since the key length of DES is only 56 bits, and people pick weak passwords are, subject to guessing attacks.

4. Passwords are not protected:
   - From eavesdropper listening to the communication
   - From spoofing program that prompts the user for password while disguising as the system login utility.
S/Key: Hash Chain (One-time Passwords)

- **Goal:** login process secure against:
  - Exposing server password file
  - Eavesdropping on login communication
  - Spoofing login program or guessing passwords.

- **Proposed by Lamport at 1981, deployed in 1994.**

- **Init:** select random $x$ (pw), compute
  $$h_1 = h(x), \ h_2 = h(h(x)), \ldots \text{ till } h_n = h^n(x).$$

- Server initially knows $h_n$

- Before $i^{th}$ login, server knows $h_{n-i+1}$

- At $i^{th}$ login, user reveals $i$

- Server confirms $h_{n-i+1} = h(h_{n-i})$
S/Key weaknesses

- User-selected passwords subject to dictionary attacks
- Not secure against active attacks
  - E.g. Server/Eve sending high login number
- Adversary catching the identifier (one time password) may use it (possibly later).
- No protection after login process.
  - S/key does not set up session key.
- Other login protocols use shared secret (e.g. password) as key – without sending it.
Questions:

1. Show a OWHF and distribution of passwords s.t. both Unix and S/Key fail.
2. A proposal is made to compute $pw' = f(host, pw)$ and use it instead of the password – evaluate.
Properties of Crypto-Hash Functions

- Randomness (uniform output)
- Confidentiality: hide the pre-image
  - Intuition: can’t find $x$ from $h(x)$
- Integrity: Collision-resistance
  - Intuition: can’t find $x, y$ s.t. $h(x) = h(y)$
  - Use: signing $h(x)$ instead of signing $x$
Collision Resistance

**Simplified (Strong) Collision Resistance Assumption:** assume that it is hard (infeasible) to find a collision, i.e. \(<x, x'>\) such that \(x \neq x'\) yet \(h(x) = h(x')\).

- Natural definition, but problematic:
  - \(h\) is fixed
  - Adversary can simply output a specific collision in it.
  - Possible fix: (public) key

- Holds for random function (oracle)
Weak CRHF

**Weakly Collision Resistant Hash Function:** it is hard to find a collision with a specific (random) $x$.

A function $h$ is a **Weakly CRHF** if:
- for every length $l \geq n$,
- given $x \in_R \{0, 1\}^l$,
- it is infeasible to find $x' \neq x$ s.t. $f(x') = f(x)$.

Property also called **2nd pre-image resistance**.
OWHF vs. (weak) CRHF

- *Is every OWHF also (weak) CRHF?*
  - No. Let $f$ be a OWHF and $g(x || b) = f(x)$. There are collisions in $g$, but it is a OWHF (inverting it clearly allows inverting $f$).

- *Is every (weak) CRHF also a OWF?*
  - *Exercise…*
Applying Weakly CRHF

- Weakly Collision Resistant Hash Function: it is hard to find a collision with a specific (random) $x$.
- Uniformly distributed input (not chosen by Adversary!)
- Alice sends message to Bob, and signs its hash
  - Bob knows that Alice sent the message
    - Only if the message is uniformly distributed!
  - Can Bob prove Alice sent (signed) the message?
Weakly CRHF may be too weak...

- Sending signed agreement:
  - Alice reaches agreement with Bob
  - Alice signs hash of agreement
  - Bob can verify Alice signed the agreement

- But: agreement *not* uniformly distributed!
  - Maybe Bob/Alice chose it to have collision?

- Solutions:
  - Signer ensures contract is `randomized` (possibly use hash with random public key)
  - Signer responsible for any properly signed version
    - But in reality both parties often influence contract
  - *Strongly/Any Collision Resistant Hash Function* (Keyed)
    - Or: keyless hash with `Simplified (Strong) Collision Resistance Assumption`
Keyed CRHF

*(Strongly/Any) Collision Resistant Hash Function (CRHF):* a set \( \{h_k\} \) of hash functions, such that any adversary, given \( k \in_R \{0,1\}^n \), cannot efficiently find collision, i.e. \( <x,x'> \) such that \( x \neq x' \) yet \( h_k(x) = h_k(x') \).

Adversary does not know \( k \) in advance – cannot prepare a collision

How many trials do we need to make to find collisions by `exhaustive search` (guessing)?
Finding Collisions – Birthday Paradox

- Compute hashes of $2^{2n/2}$ random values
- With probability over $\frac{1}{2}$, there will be a collision
- Why? - `birthday paradox` (in probability course)
  - Intuition: probability of a collision to given $x$ is roughly $1/2^n$; but we allow any collision

- Conclusion: for collision resistance we need *double* the `effective key length`

- In practice: searching $2^{64}$ values required one month with 10M$ machine in 1994 [OW94]
  - Expected cost today: less than 100,000$

→ Use longer block (output) size (e.g. 160bits)
→ Try not to require (strong) collision resistance
Target-Collision Resistant Hash

Target-Collision Resistant (TCR) Hash Function: a set \( \{ h_k \} \) of hash functions, such that any adversary, and any \( x \), given \( k \in_R \{0,1\}^n \), cannot efficiently find a collision \( x' \) such that \( x \neq x' \) yet \( h_k(x) = h_k(x') \).

Adversary (Bob) must pick collision \( (x) \) in advance (before knowing \( k \))!

Alice selects \( k \); she can safely use Bob’s suggested contract...

Also called: Universal One-Way Hash Functions (UOWHF)

Sufficient for signing: \( h_k(x) \) identifies \( x \)
Other Collision Resistance Properties

- **Correlation (Near Collision) Resistant Hash Function**: hard to find \( <x, y> \) s.t. \( h(x), h(y) \) differ only in \( k \) (a small number of) bits.

- **Relation Resistant Hash Function**: hard to find \( <x_1, x_2, \ldots> \) s.t. \( \text{Relation}(h(x_1), h(x_2), \ldots) = \text{True} \), where \( \text{Relation} \) is some predicate. For example:
  - Negation: \( h(x) \) is the complement of \( h(y) \).
  - Sum: \( h(x) + h(y) = h(z) \)

- See motivation and definitions in [Anderson 93].
- Lesson: identify necessary properties!
- Or… use random oracle analysis…
Designing CRHF

- Problem: Variable Input Length (VIL)
  - Hard to design and test (by cryptanalysis)
  - Idea: build VIL CRHF from FIL CRHF
  - FIL CRHF are also called compression function: \( \text{comp} : \{0,1\}^{2n} \rightarrow \{0,1\}^n \)

\[
\begin{align*}
x \in \{0,1\}^n & \quad \text{comp} \quad \text{comp}(x,y) \in \{0,1\}^n \\
y \in \{0,1\}^n &
\end{align*}
\]
Designing Keyed CRHF

- Build keyed CRHF from keyed compression (FIL) function:
  \[ c_k : \{0,1\}^{2n} \rightarrow \{0,1\}^n \]

- We ignore the key in the next few foils

\[ x \in \{0,1\}^n \rightarrow c_k c_k(x,y) \in \{0,1\}^n \]

\[ y \in \{0,1\}^n \rightarrow \]

Advanced!
Constructing VIL CRHF from FIL CRHF

- Idea: use iterative process, compressing block by block
- Let the input $x$ be $l$ blocks of $n$ bits
  - Pad the last block (with 0’s) if necessary [how to remove? later…]
- Let $y_0=IV$ be some fixed/random $n$ bits (IV=Initialization Value)
- For $i=1,..l$, let $y_i=c(x[i], y_{i-1})$; output $h(x)=y_{l+1}$
- Prefix attack: Pick prefix $p$ and random $IV=v$. Let $z=h_v(p)$ with $IV=v$. Then for any $x$ holds: $h_z(x)=h_v(p||x)$.
  - Exercise (*): prefix attack with fixed IV for a FIL CRHF $c$

![Diagram of iterative process](image-url)
Merkle-Damgård FIL$\rightarrow$VIL Hash

- Build $h$ from compression function: $c : \{0,1\}^{2n} \rightarrow \{0,1\}^n$
- Let the input $x$ be $l$ blocks of $n$ bits
  - Pad the last block if necessary
  - Add extra block, $x[l+1]=|x|$ (before padding – so we can remove it)
- Let $y_0=IV$ be some fixed $n$ bits (IV=Initialization Value)
- For $i=1,..l+1$, let $y_i=c(x[i],y_{i-1})$
- Output $h(x)=y_{l+1}$

\begin{center}
\begin{tabular}{l|l|l|l|l}
\hline
IV  & $c$ &  & $c$ &  \\
\hline
\hline
& $c$ &  & $c$ &  \\
\hline
\end{tabular}
\end{center}

$h(x)=y_{l+1}=c(|x|,y_l)$

Idea: given $h(x)=h(x')$, for $x \neq x'$, we can find $z \neq z'$ s.t. $c(z)=c(z')$. 

11/26/2003

http://Amir.Herzberg.name
Theorem (simplified): if $c$ is collision-resistant, then $h$ is collision resistant.

Proof (sketch): we use collision in $h$ to find collision in $c$. Suppose $h(x) = h(x')$ for $x \neq x'$.

- $h(x) = c(|x| \parallel y_{|x|}) = c(|x'| \parallel y'_{|x'|})$. Hence assume $|x| = |x'|$ and $y_{|x|} = y'_{|x'|}$ (or collision in $c$).
- Recursively for $j = l$ to 1, we have $y_j = y'_j$, i.e. $c(x[j] \parallel y_{j-1}) = c(x'[j] \parallel y'_{j-1})$. Hence $x[j] = x'[j]$ and $y_{j-1} = y'_{j-1}$. But $x \neq x'$!
Alternative - Hash Trees

- To hash a long document or many docs...
  - Hash each document (or part)
  - Hash all hashes (possibly recursively)
  - Can use compression function(s) (with finite input)
- Less efficient than MD when validating all inputs
- Requires to keep state (logarithmic in document size)
- Advantages when validating only some inputs:
  - Efficiency: validate only what you need
  - Reuse: some recipients may not need all docs
  - Privacy: some docs may not be shared with all

\[ h(h(Doc1))...h(Doc5)) \]

\[ h(Doc1) \]
\[ h(Doc2) \]
\[ h(Doc3) \]
\[ h(Doc4) \]
\[ h(Doc5) \]

Doc1  Doc2  Doc3  Doc4  Doc5
Properties of Crypto-Hash Functions

- Randomness (uniform output)
- Confidentiality: hide the pre-image
  - Intuition: can’t find $x$ from $h(x)$
- Integrity: Collision-resistance
  - Intuition: can’t find $x, y$ s.t. $h(x) = h(y)$
  - Use: signing $h(x)$ instead of signing $x$
- Multiple properties
  - `Standard`, multi-purpose hash functions
Hash with multiple properties

- We saw multiple goals/definitions for crypto-hash functions:
  - Confidentiality properties, e.g. OWHF
  - Randomness properties
  - Collision resistance properties
- All properties exist for ‘random oracle’
- Needed:
  - standard, `general-purpose` cryptographic hash
    - A basic ‘crypto building block’ (like block cipher / PRP)
    - Specific choices of ‘seemingly random’ functions
    - Why not use a pseudo-random function (PRF)?
  - Cryptanalysis tolerance – combining proposals
    - Like cascade for PRPs (block cipher)
Cryptanalysis-tolerant Hash: Cascade

- Construct $h$ by composing $h_1$, $h_2$
- Cascade composition: $h(x) = h_1(h_2(x))$.
- Clearly fails for `very weak` $h_1$, $h_2$
- Example: $h_1(x) = 0 \Rightarrow h(x) = h_2(0)$
- Assume $h_1$, $h_2 : \{0,1\}^* \rightarrow \{0,1\}^L$ are regular:
  - For every $l > L$, $y, y' \in \{0,1\}^L$, the number of pre-images of length $l$ of $y$ and $y'$ is (almost) equal
- Cascading of regular functions ensures cryptanalysis-tolerance for OWF property:
  - If one of $h_1$, $h_2$ is one-way function, then $h$ is OWF
- But… any collision of $h_2$ is a collision of $h$
Cryptanalysis-tolerant Hash: Parallel Composition

- Parallel Composition: \( h(x) = h_1(x) \parallel h_2(x) \)
- Claim: collision for \( h \) \( \Rightarrow \) collisions for both \( h_1 \) and \( h_2 \)
- Proof: suppose \( h(x) = h(x') \), i.e. \( h_1(x) \parallel h_2(x) = h_1(x') \parallel h_2(x') \). Hence \( h_1(x) = h_1(x') \), \( h_2(x) = h_2(x') \).
- Holds also for keyed (Any/Target CR) hash
  - If using different key for each hash! (Why??)
- \( \Rightarrow \) If either \( h_1 \) or \( h_2 \) is a weak CRHF, TCRHF or ACRHF, then \( h \) is a weak CRHF, TCRHF or ACRHF, respectively.
- But parallel composition is bad for confidentiality
  - \( x \) `more exposed`
  - E.g. if \( h_1 \) not OWHF than \( h \) is not OWHF…
Cryptanalysis-tolerant Hash: cascading with input

- Cascade $h(x) = h_1(h_2(x))$: easier to find collisions…
- Parallel $h(x) = h_1(x) || h_2(x)$: easier to find pre-image
- What about cascading with input: $h(x) = h_1(x \ || \ h_2(x))$?
  - A pre-image of $h()$ provides a pre-image of $h_1$
  - Collision in $h()$ implies collision in $h_1$
  - Assuming only few collisions in $h_1$, say $h_1(x||y) = h_1(x'||y')$… Requires $y' = h_2(x')$, $y = h_2(x)$ (unlikely – at least if $h_1, h_2$ were random)

- This construction offers some confidentiality and some collision-resistance properties…
- Used in several `standard` hash functions…
Several hash standards are widely-used standards:

- Allowing security by evidence of failed cryptanalysis
- Many efficient, free/inexpensive, interoperable implementations
- All existing standards are un-keyed crypto hash functions:
  - MD5 (MD = Message Digest)
  - SHA-1 (SHA = Secure Hash Algorithm)
  - RIPEMD

Stated Goals:

- Collision-Resistance: `strong CRHF` and `weak CRHF`
- Confidentiality: one-way function

All are very efficient, e.g. cf. to encryption

All use Merkle-Damgård iterative construction +…
MD5: Compressing block $i$

Developed by RSA Inc.
Output and chaining-value are four 32-bit words (128 bits)

$x[i]$ is 16 words (32 bits each) $\rightarrow$ 512b

Input in 512b blocks

Not in MD construction!
goal=output distribution?

Addition mod $2^{32}$

Cascading with input
MD5 Compression Functions

- All four functions \(c_1, \ldots, c_4\) have same structure.
- Break 128b `chaining value` \(Y[i]\) to four 32-bit words: A, B, C, D.
- Each function has 16 rounds \(r=1..16, \ldots 64\).
- Single round computation:
  - \(A_{r+1} = D_r, \quad C_{r+1} = B_r, \quad D_{r+1} = C_r\)
  - \(B_{r+1} = B_r + \ll_s s[r] (A_r + g(B_r, C_r, D_r) + x[i][r] + T[i])\)
  - \(T[i] = \text{int}(2^{32} \text{ abs(sin(i)))}\)
  - \(\ll_s\) is circular left shift by \(s\); \(s[r]\) is a fixed table.
- No published analysis / proof.
Collisions in MD5

- Output is 128 bit
  - Collisions can be found with $2^{64}$ time and storage
  - Feasible?
- Collisions found in the compression function
  - But MD5 uses a specific Initial Value – so the collision found are not (yet?) for MD5
- Still widely used, but being `phased out`
- About twice faster than RIPE-MD, SHA-1
SHA-1 (Secure Hash Algorithm)

- Developed by NIST, published as FIPS 180-1
- Output is 160 bit
  - New versions: 256b, 384b and 512b proposed
- Widely used; `closed` design process, criteria
- Very similar design to MD5
  - 160b chaining block
  - Chaining value added (mod $2^{32}$) to output of compression
RipeMD-160

- Developed by EU RICE project
- Open design process, criteria
- Variants: 128, 160, 256 or 320 bits
- RIPEMD-160 most common
- Compression function:
  - Is RipeMD OWF, assuming one/few blocks are OWF?
  - Same for collision-resistance
Towards Cryptanalysis-tolerant Hash

- Goal: cryptanalysis-tolerant confidentiality and collision resistance
- 1st idea: combine parallel and serial compositions:

Confidentiality (OWF): Ok for regular functions (cascade).

Collision-resistance: No
Select some \( m \neq m' \).
Select \( h_0 \) s.t.:
\[
\begin{align*}
    h_0(m) &= h_0(m') \\
    h_0(h_1(m)) &= h_0(h_1(m'))
\end{align*}
\]

http://Amir.Herzberg.name
The D Cryptanalysis-tolerant Hash Construct

- **Idea:** use four different candidate hash functions:
  \[ h(m) = h_1(h_0(m)) \parallel h_3(h_2(m)) \]

- **h** is OWF if any three of the candidates are OWFs
  - Holds for other confidentiality properties as well

- **h** is weakly CRHF if any three of the candidates are weakly CRHF (and all four are regular)

- With (independently) keyed hash functions: **h** is ACR/TCR hash if any three of the candidates are ACR/TCR hash (respectively)

- Can we require less than \( \frac{3}{4} \) ‘good’ functions?

```
11/26/2003
http://
```
The E Cryptanalysis-tolerant Composition

- Combine *three* functions: \( E[h_0, h_1, h_2] \)

- Confidentiality: Ok
- Collision-resistance: Ok

Why? Collision of \( E \to h_o(h_1(m)) = h_0(h_1(m')) \)

Collision of either \( h_o \) or \( h_1 \)

- Assuming \( h_o, h_1, h_2 \) are *all* regular functions
- Can we avoid this assumption? ... see paper
Crypto-Hash functions are useful for

- Providing short `digest` of long documents
- Extracting randomness
- Confidentiality: hiding pre-image (original document)
- Integrity: detecting changes
- Proving knowledge of pre-image

Be careful in definition/assumption used

- One-way property may expose some (of the) input
- Random oracle analysis – simple argument of security
- Prefer cryptanalysis-tolerant constructions