DIMACS Security & Cryptography
Crash Course – Day 1
Hashing

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

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

- Crypto-Hash properties
- Using and Collecting Randomness
- Randomness of Hash
- Confidentiality of Hash
- One-way functions
- Random Oracle
- Integrity & Collision Resistance

- 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
Collecting Randomness

- Use available sources with some randomness
  - Different `unpredictable, unobservable` events
- Extract random seed \( n \) \( \text{bits} \)
  - In practice: usually using `cryptographic hash function`
- Use PRG to generate sufficient random bits
- Certainly Ok if hash was a random function…
Random Oracle Methodology

- Analyze as if hash $h()$ 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 a good argument/evidence of security.
Confidentiality of Hash

- Hash has no secret key
  - Cannot use to send secret message
- But hash should hide input
  - Cannot learn input given output (`one way function`)
- $f$ is OWF (One Way Function) if:
  - $f$ is computed by some PPT algorithm,
  - yet for any PPT alg. $A$: $P_A(n) = \Pr\{f(A(f(x))) = f(x) : x \in \{0,1\}^n\} \approx p_0$
- PPT: Probabilistic Polynomial Time algorithm
  - Time complexity $< p(n)$ for some polynomial $p(n)$
  - $P_A(n) \approx_p 0$:
    - Every polynomial $p(n)$, exists some $l_{min}$ s.t. if $n > l_{min}$ and $x \in \{0,1\}^n$ then $P_A(n) < 1/p(n)$.
- Asymptotic definition; says nothing about any fixed input length
- Worse – maybe $f$ exposes partial info on input?
- Most works use `random oracle` to simplify security analysis
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 a 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 \( x \in \{0,1\}^l \),
  - it is infeasible to find \( x' \neq x \) s.t. \( f(x') = f(x) \).

- Property also called **2nd pre-image resistance**.
### 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)
    - Or: keyless hash with `Simplified (Strong) Collision Resistance Assumption`
  - Signer responsible for any properly signed version
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: $\text{comp} : \{0,1\}^{2n} \rightarrow \{0,1\}^n$

\[
\begin{align*}
{x} & \in \{0,1\}^n & \text{comp} & \rightarrow & \text{comp}(x) & \in \{0,1\}^n \\
{y} & \in \{0,1\}^n
\end{align*}
\]
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 if necessary
- 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: $h_z(x) = h_v(p || x)$.

```
x[1] x[2] ... x[l]
```

```
IV   c   c   c
```

$h(x) = y_{l+1} = c(x[l], y_{l-1})$
Merkle-Damgard FIL→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, $x[l+1] = |x|$
- 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 $h(x)=y_{l+1}$

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

\[
\begin{array}{cccccc}
IV \rightarrow & c & c & & & c \rightarrow h(x)=y_{l+1}=c(|x|,y_l) \\
\end{array}
\]
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 for unkeyed 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-Damgard iterative construction +…
Conclusion

- 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
Extras...
Finding Collisions – Birthday Paradox

- Compute hashes of \(2^{2^n/2}\) random values
- With probability > \(\frac{1}{2}\), there will be a collision
- Why? - `birthday paradox`(Proof omitted)
  - 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$
- Consider weaker notions
Security of MD Construction

Theorem: if \( \text{comp} \) is collision-resistant, then \( h \) is collision resistant.

Proof: we use collision in \( h \) to find collision in \( \text{comp} \). Suppose \( h(x) = h(x') \) for \( x \neq x' \).

- Denote \( l = |x| \); note \( x[i+1] = l \). Hence
  \( h(x) = \text{comp}(l || y) = \text{comp}(l' || y') \). Hence assume \( l = l' \) and \( y_i = y'_i \) (or collision in \( \text{comp} \)).

- Recursively for \( j = l \) to \( 1 \), we have \( y_j = y'_j \), i.e.
  \( \text{comp}(x[j] || y_{j-1}) = \text{comp}(x'[j] || y'_{j-1}) \).
  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

\[
h(h(Doc_1)|...|h(Doc_5))
\]
Hash with multiple properties

- We saw multiple goals/definitions for crypto-hash functions:
  - Confidentiality properties, e.g. OWHF
  - Randomness properties, e.g. $t$-resilient PR hash
  - Collision resistance properties: weak CRHF, $t$-resilient

- Goals:
  - Hash satisfying multiple goals
    - To have standard, `general-purpose` crypto-hash
Cryptanalysis-tolerance: Cascade

- Construct $h$ by composing candidates: $h_1, h_2, \ldots$
- 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 confidentiality:
  - If one of $h_0, h_1$ is one-way function, then $h$ is one-way
- But… any collision of $h_2$ is a collision of $h$
Parallel Composition

- Parallel Composition: \( h(x) = h_1(x) \ || \ h_2(x) \)
- Claim: collision for \( h \) \( \rightarrow \) collisions for both \( h_1 \) and \( h_2 \)
- Proof: suppose \( h(x) = h(x') \), i.e. \( h_1(x) \ || \ h_2(x) = h_1(x') \ || \ h_2(x') \). Hence \( h_1(x) = h_1(x') \), \( h_2(x) = h_2(x') \).

- If either \( h_1 \) or \( h_2 \) is a (weak / \( t \)-resilient) CRHF, then \( h \) is a (weak / \( t \)-resilient) CRHF.

- But parallel composition is **bad for confidentiality**
  - \( x \) `more exposed`
  - E.g. if \( h_1 \) not OWHF than \( h \) is not OWHF…

- We often require hash with *multiple properties*
`Hybrid` composition...

- 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 $y' = h_2(x')$, $y = h_2(x)$
- This construction offers some confidentiality and some collision-resistance properties...
- Used in `standard` hash functions MD5, SHA-1...
Merkle-Damgard + Partial Regularity

- MD construction: Build $h$ from compression function: $c : \{0,1\}^{2n} \rightarrow \{0,1\}^n$
- Let the input $x$ be $l$ blocks of $n$ bits
- Let $y_0 = IV$ be some fixed $n$ bits (IV=Initialization Value)
- Partial regularity: if IV is uniformly-distributed, then so is $h(x)$
- How? For $i = 1,..,l+1$, let $y_i = y_{i-1} + c(x[i], y_{i-1})$
- Output $h(x) = y_{l+1}$

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

![Diagram](attachment:image.png)

$IV \xrightarrow{c} c \xrightarrow{c} h(x) = y_{l+1} = c(|x|, y_l)$
MD5

- Developed by RSA Inc.
- Output is 128 bit
  - Collisions can be found with $2^{64}$ time and storage
  - Believed feasible (with about 100,000$ equipment for 1 month)
- Collisions found in the compression function
  - But only in the chaining value – so not a collision for MD5 (yet)
- Still widely used, but being `phased out`
- About twice faster than RIPE-MD, SHA-1
- Compression function: Cascade of four 128b+512b → 128b compression functions
MD5: Compressing block $i$

$y[i]$ is added to $c_1$, $c_2$, $c_3$, and $c_4$ with addition mod $2^{32}$.

$x[i]$ is 16 words (32 bits each) → 512b
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:
  - $A_{r+1}=D_r$, $C_{r+1}=B_r$, $D_{r+1}=C_r$
  - $B_{r+1}=B_r+<_{s[r]} (A_r+g(B_r,C_r,D_r)+x[i][r]+T[i])$
  - $T[i]=\text{int}(2^{32} \text{ abs}(\sin(i)))$
  - $<_{s}$ is circular left shift by $s$; $s[r]$ is a fixed table
- No theory behind design, no analytical proof
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:** provably cryptanalysis-tolerant hash
- **1st idea:** combine parallel and serial compositions:

  Confidentiality: Ok for regular functions (cascade).

  Collision-resistance: No

  Select some \( m \neq m' \).

  Select \( h_0 \) s.t.:
  
  \[
  h_0(m) = h_0(m') \\
  h_0(h_1(m)) = h_0(h_1(m'))
  \]
The *E* Cryptanalysis-tolerant Composition

- **Goal:** provably cryptanalysis-tolerant hash
- **2nd idea:** combine *three* functions: $E[h_0, h_1, h_2]$

- Confidentiality: Ok
- Collision-resistance: Ok

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

Collision of either $h_o$ or $h_1$

- Assuming $h_0$, $h_1$, $h_2$ are *all* regular functions
- Can we avoid this assumption? … see paper
Recall `paper, stone, scissors`

- **Confidentiality**
  - Bob can’t know what Alice chose

- **Collision-resistance**
  - Alice can’t `change her hand`

- **Randomness**
  - $h(x)$ appears `random`
  - If $h(x)$ is deterministic, confidentiality
Commitment Schemes

- Commitment ≈ Collision resistance + privacy
- Three functions: Commit, Decommit, Validate
  - Commit, Decommit have two inputs: message, random
  - Validate(m,Commit(m,r),Decommit(m,r))=True
- Security properties
  - Confidentiality: Commit(m,r) reveals nothing about m
  - Collision-resistance: infeasible to find m, m’, d, d’, c s.t. Validate(m,c,d)=Validate(m’,c,d’)=True
- Unfortunately this is impossible…
Randomness Required for Collision Resistance

- Collision-resistance: infeasible to find $m, m', d, d', c$ s.t. $\text{Validate}(m,c,d) = \text{Validate}(m',c,d') = \text{True}$

- But: for any \textit{Commit} function there exist collisions: $<m,r>, <m',r'>$ s.t. $c = \text{Commit}(m,r) = \text{Commit}(m',r')$

- So maybe Alice knows such collision?
  - And then: $\text{Validate}(m,c,d) = \text{Validate}(m',c,d') = \text{True}$ where $d = \text{Decommit}(m,r)$, $d' = \text{Decommit}(m',r')$

- Solutions:
  - Use \textit{keyed commit function} with random (public) key
  - Or: ensure input to commitment is randomized
  - Recipient confirms proper randomization

- Still need random $r$ for each new commitment!
Keyed Commitment Schemes

- **Keyed functions**: Commit, Decommit, Validate
- Commit<sub>k</sub>, Decommit<sub>k</sub> have inputs: key k, message, random
- Validate<sub>k</sub>(m, Commit<sub>k</sub>(m,r), Decommit<sub>k</sub>(m,r))=True
- Confidentiality: Commit<sub>k</sub>(m,r) reveals nothing on m
- Collision-resistance: no adversary ADV, given random k, can efficiently find m, m’, d, d’, c s.t.
  Validate<sub>k</sub>(m,c,d)=Validate<sub>k</sub>(m’,c,d’)=True
- Recipient confirms k is random, not chosen by ADV!
- If recipient adds randomness, we can avoid key!
Interactive Commitment Schemes

- Receiver (Bob) selects random input \( r_B \)
- Three functions: \( \text{Commit}, \text{Decommit}, \text{Validate} \)
  - \( \text{Commit}, \text{Decommit} \) have three inputs: message, \( r_A, r_B \)
  - \( \text{Validate}(r_B, m, \text{Commit}(m, r_A, r_B), \text{Decommit}(m, r_A, r_B)) = \text{True} \)
- Security properties
  - Confidentiality: \( \text{Commit}(m, r_A, r_B) \) reveals nothing about \( m \)
  - Collision-resistance: no adversary \( \text{ADV} \), given random \( r_B \), can efficiently find \( m, r_A, m', d' \) s.t.
    \( \text{Validate}_k(r_B, m', \text{Commit}_k(m, r_A, r_B), d') = \text{True} \)
`Paper, stone, scissors` using Interactive Commitment Scheme

Bob

Ladies first…
Please use $r_B$

commit($Paper, r_A, r_B$)

Stone

Alice

Paper, $r_A$

You won!

Decommit is often trivial
Commitment from Hashing

- `Standard` construction in practice:
  - \( \text{Commit}(m,r_A,r_B) = h(m||r_A||r_B) \)
  - \( \text{Decommit}(m,r_A,r_B) = r_A \)
  - \( \text{Validate}(r_B,m,c,d) = \text{TRUE if } c = \text{Commit}(m,d,r_B) \)

- Justified by:
  - Random oracle analysis, or ??? (ongoing work)

- Other provable-secure constructions require weaker \( h \)
  - But are more complex, not used in practice
  - Only keyed versions
  - Much theory work, e.g. zero-knowledge proofs,…
Application: Secure Government Bid

Goals:
- Receive `sealed bids` until deadline
- Open all bids, select the best after deadline

Concerns:
- Leakage of info about bids to other bidders
- Changing of bid after deadline

Solution:
- Publish RFP with randomizer $r$
- Bidders send $h(bid, r, r')$
- At deadline, government publishes all commitments to bids
- Then participants publish $bid$ and $r'$