Indigestion: Assessing the impact of known and future hash function attacks

Eric Rescorla
Network Resonance
ekr@networkresonance.com

Overview

- Hash functions are used all over cryptographic protocols
- Only a few common functions
  - MD5, SHA-1, SHA-256, SHA-384, SHA-512
  - Only MD5 and SHA-1 are commonly used
  - No good theoretical foundation for constructions
- First good attacks on MD5 and SHA-1 published in 2004-2005
- Impact of current attacks on our protocols?
- Impact of plausible future attacks?
- What's being done about it?

Review of hash function terminology

Collision Find M, M' st $H(M) = H(M')$

1st preimage Given X, find M st $H(M) = X$

2nd preimage Given M, find M' st $H(M') = H(M)$

In a perfect hash function of length l:

- Collisions require $2^{l/2}$ effort to find
- 1st and 2nd preimages require 2^{l} effort to find

The Current Situation

MD5 Collisions can be easily found [Wang et al.][Klima06] "Target" collisions can be found with 2^{52} effort [Stevens et al.].

SHA-1 Collisions in SHA-1 with 2^{63} effort (design goal $= 80$ bits) [Wang et al.]

Certificates Lenstra et al. demonstrate a pair of certificates with different public keys but the same hash (and hence signature)

HMAC Theoretical forgery attacks [Kim et al.]

Important limitations:

- None of these attacks allows you to compute a preimage
- The colliders are not totally controllable

How hash functions are used

- Digital signatures
- Key derivation (PRFs, KDFs)
- MACs
- Data fingerprinting (e.g., Tripwire)

The Wang collision attack

- Start with an arbitrary hash state (IV)
- Compute four blocks M_0, M_1, M_0', M_1' st.
  - $H(M_0, M_1) = H(M_0', M_1')$
- Relationships between colliders are not free
  - Some bits must be the same, some different, some arbitrary, some fixed
- How practical is this?
  - Very for MD5: Klima shows 17 seconds on a 3.2 GHz P4
  - Not so for SHA-1: 2^{63} operations
    - $*$ No collision published yet for SHA-1

The Stevens target collision attack

- Start with two messages M_a, M_b
- Compute two suffixes S_a, S_b st.
  - $H(M_a||S_a) = H(M_b||S_b)$
- How practical is this?
  - Suffixes are long (4000 bits)
    - * And random-looking
  - Work factor is about 2^{52} (for now)

Overview

- Hash functions are used all over cryptographic protocols
- Only a few common functions
  - MD5, SHA-1, SHA-256, SHA-384, SHA-512
  - Only MD5 and SHA-1 are commonly used
  - No good theoretical foundation for constructions
- First good attacks on MD5 and SHA-1 published in 2004-2005
- Impact of current attacks on our protocols?
- Impact of plausible future attacks?
- What's being done about it?

Essential Elements of a Collision Attack

1. Need a three party situation
    - Alice generates a message
    - Bob (and maybe Alice) signs it
    - Carol acts upon it

Proposed Uses of Collision Attacks

- Contract signing (classic example)
- Forging certificates
- Compromised software distribution [Kaminsky]
- Fake authorizations [Daum and Lucks]

Contract Signing and Collisions

- Attacker generates two variants
  - M1 = "I will pay Eric $10.00/hr" (a bargain)
  - M2 = "I will pay Eric $10000/hr" (a rip-off)
- Attacker gets victim to sign M1 (E.g., in an S/MIME message)
- Then claims victim signed M2
  - And he has evidence to prove it
- Small problems
  - Real contracts don't have random garbage
  - Victim has both variants
- Big problem
  - This isn't how contracts actually work

Contracts in the Real World

- You and I negotiate a contract
  - Your lawyer sends me the final copy
  - I sign the last page
  - I fax it over to you
  - You fax it back
- No attempt is made to bind contents to signature
  - At most, I might initial each page
  - But sometimes, just last page is exchanged!
- Signature is unverified
  - How does relying party know, anyway?
  - An "X" can be binding!
- It's the intention that counts

Essential Elements of a Collision Attack (Revised)

1. Need a three party situation
   - Alice generates a message
   - Bob (and maybe Alice) signs it
   - Carol acts upon it
2. *Signature needs to be mechanically evaluated*
3. *Find somewhere to hide the random junk*

Software Distribution

- Many software distributions use hashes for integrity checking
- Generate a patch for a Foolix 1.0 in two versions
  - One innocuous
  - One containing a remote exploit
- When Foolix 1.1 is released I swing into action
  - Break into Foolix's distribution server
  - Replace the good version with my version
  - ...
  - Profit
- That sounded a lot better in my head...

Problems with Software Distribution Attacks

- Hard to predict contents of binaries (prefix)
  - Compilation timestamps
  - Especially if other people are checking in stuff
- Somewhat easier on source, but...
  - Other people still might check in
  - Plus CVS/SVN timestamps, Id values, etc.
- Still need to somehow replace the "good" copy of the program
  - Theoretically easier if program is P2P-distributed
  - ... Unless that distribution includes its own checksums
- Why go to all this trouble?
  - Most programs aren't that well reviewed anyway
  - How hard is it to insert vulnerabilities anyway?

Impact of a single collision

- Collisions are easy to generate with MD5
- But expensive with SHA-1 (2^{63})
  - ... none have been generated yet
  - Eventually one will be as a demonstration
- What can you do then?
  - Use it as a binary switch
  - Daum and Lucks show a switch-hitting postscript doc

A	B

```
if(A)
    display X
else
    display Y
```

Targeted Collisions and Certificates

- Randomized values make regular collision attacks impractical
  - Cert is packed too tightly to tweak DNs
  - Best attack is to demonstrate two certs with same DN and different public keys
- Targeted collisions are better
  - Can start with different DNs
  - And then “repair” to a collision
- This isn’t really practical
  - Requires way too many blocks
  - Too many computations
  - And knowledge of prefix values

Bottom Line: Current Status

- Not affected
  - Key derivation functions (PRFs)
  - Peer authentication without non-repudiation (SSL, IPsec, SSH, etc.)
  - Message authentication (HMAC)
  - Challenge-response protocols (probably)
- Affected
  - Non-repudiation (at least technically)
  - Certificate issuance – but only in some special cases
  - Timestamps (maybe)
- This assumes certificates remain secure

Overview

- Hash functions are used all over cryptographic protocols
- Only a few common functions
  - MD5, SHA-1, SHA-256, SHA-384, SHA-512
  - Only MD5 and SHA-1 are commonly used
  - No good theoretical foundation for constructions
- First good attacks on MD5 and SHA-1 published in 2004-2005
- Impact of current attacks on our protocols?
- Impact of plausible future attacks?
- What's being done about it?

Potential Future Attacks

- Controllable collisions
- 2nd preimage
- 1st preimage
- Compromise of HMAC
  - Forgery
  - Some kind of reversal

Controllable Collisions and certificates

- What if we could really control collisions?
- Attacker generates two names
  - Good: `www.attacker.com`
  - Bad: `www.a-victim.com`
- Sends a CSR with good name to CA
  - CA signs cert
  - Attacker now has cert with victim's name
- Two problems
  - Can you predict the prefix?
  - What about the random padding?

The structure of certificates

```
TBSCertificate ::= SEQUENCE {
        version                 Integer value=2
        serialNumber            Integer (chosen by CA)
        signature               algorithm identifier
        issuer                  CA's name
        validity                date range
        subject                 subject's name
        subjectPublicKeyInfo    public key
        extensions              arbitrary stuff
}
```

- The signature is over $H(TBSCertificate)$

Prefix prediction

- Knowing which values to use depends on the prefix
  - But the prefix isn't totally fixed
  - This is a total design accident!
- All but serial number and validity are fixed
  - Sequential serial numbers are easy to predict
    * At least to within a few
    * Verisign uses $H(time\_us)$ which is hard to predict
  - How quantum is the validity?
    * Verisign seems to use a fixed "not before" but a "not after" based on the current time
    * So predictable to within a few hundred seconds?
- Attacker is likely to need to try the attack a number of times
- Randomizing serial number is a simple countermeasure

A vulnerable certificate structure

```
TBSCertificate ::= SEQUENCE {
         version                Integer value=3
         signature              algorithm identifier
         issuer                 CA's name
         subject                subject's name
         subjectPublicKeyInfo   public key
         serialNumber           Integer (chosen by CA)
         validity               date range
         extensions             arbitrary stuff
}
```

Hiding the randomness

- Remember, we want a specific target name
  - E.g. `www.amazon.com`
  - Though we have flexibility in the name we send the CA
- Random padding can be concealed in pubkey
  - Remember, modulus doesn't have to be p*q
  - As long as we can factor it
    - ... which is likely for a random modulus [Back 04]

2nd Preimage Attacks

- Attack description:
  - Given some message M
  - Find some message M' st H(M) = H(M')
- Classic example: message forgery
  - Start with signed "Good" message
  - Transform it into signed "Bad" message

2nd preimages and certificates

- This is really serious
  - Attacker should be able to forge a cert of his
  - Validity of all certs with this digest would be questionable
  - Say goodbye to any cert-based protocol
  - No useful countermeasures
- How likely do we think this is with MD5?
  - If so, really bad
  - Lots of valid certificates use MD5!
- SHA-1 comfort level is higher

2nd preimages and other protocols

- Remember: three major uses of hashes
  - MACs
  - Key expansion
  - Signatures
- Only signatures are threatened
- But they're commonly used
  - SSH, SSL, IPsec key agreement
    * Signatures are over nonces
    * Only works if very fast (need to beat timeouts)
  - S/MIME authentication
- And of course all but SSH depend on certificates
- So, this is bad...

Distributed Hash Tables

- DHTs use hashes a lot
  - Values typically stored as $H(lookup\_key)$
    - Sometimes $H(Value)$
  - Identities are often $H(IP)$
  - Hashes used for data integrity
- Collisions aren't that useful here
  - You just contaminate your own data
- Preimages let you contaminate other people's data

Compromise of HMAC

- HMAC is *the* standard MAC for most protocols
- Proofs of security [Bellare et al.]
  - But these don't apply if the interior hash function is weak
- So, what if we had a good attack?
  - Forge new messages without the key
  - Extract the key?

Impact of HMAC Forgery: SSL/TLS

- SSL/TLS uses authenticate then encrypt (AtE)
  - Send $E(K_e, Message||HMAC(K_m, Message))$
- Block ciphers
  - Can't re-insert the MAC
    - It gets randomized
  - And wouldn't match the data in any case
- Stream ciphers
  - Can reinsert MAC
  - ... but only if you know the plaintext
  - And need to know *entire* plaintext to get a match

Impact of HMAC Forgery: IPsec

- IPsec uses encrypt then authenticate (EtA)
  - Send $E(K_e, Message)||HMAC(K_m, Ciphertext)$
- Easy to do an existential forgery
  - Modify ciphertext and produce matching MAC
  - Works with block or CTR-mode ciphers
- Harder to do a targeted forgery
  - Unless you know the plaintext
    * Or part of it

Impact of HMAC Key Reversal

- Can you extract master key from HMAC-based KDF?
  - Given $X, HMAC(K, X)$, extract K
  - $|K|$ must be not too much larger than length of HMAC output
- Plausible scenarios
  - IPsec (remember, EtA)
    - Extract K_m, work backward to master key (typically DH ZZ)
      - Remember: use $H(K)$ when $|K| > blocksize$,
  - SSL/TLS (only in NULL encryption mode)
    - Extract K_m, work backward to master secret
      - This doesn't work with RSA ($|MS| = 384$ bits)
      - Plausible with DH but need to work backward to PMS
- Truncation helps here

Overview

- Hash functions are used all over cryptographic protocols
- Only a few common functions
  - MD5, SHA-1, SHA-256, SHA-384, SHA-512
  - Only MD5 and SHA-1 are commonly used
  - No good theoretical foundation for constructions
- First good attacks on MD5 and SHA-1 published in 2004-2005
- Impact of current attacks on our protocols?
- Impact of plausible future attacks?
- What's being done about it?

Transitioning to New Hash Functions

- IETF currently upgrading all protocols
- Currently available
  - SHA-256, SHA-384, SHA-512
- New but "API compatible constructions"
  - Just define new code points and drop in
  - Hopefully these will come out of NIST competition
- Incompatible constructions
  - Randomized hashes
    - Where do we put the randomizer?
  - ...

Transition Strategy Goals

- **Backward compatibility**
  - Switch-hitting can speak to old
  - New can speak to switch-hitting
- **Newest common version**
  - Two switch-hitting implementations should use the new version
- **Downgrade protection**
  - Attacker can't force you to use a weaker algorithm
- Don't have to change protocol structure again for newer algorithms
- Diversity in certificates makes this harder

S/MIME Transition Issues

- Major problem is first message (if signed)
- Assume sender has two certificates
  - Otherwise there's no choice
- Sender has no information about receiver's capabilities
  - Either certificate is potentially wrong
  - Have to guess based on overall upgrade rate
- Why not sign twice (once with each certificate)
  - Some implementations don't like partially verifiable messages
  - S/MIME standard wasn't totally clear here
    - Clarification in RFC 4853 [Housley]
- Subsequent messages can use capabilities attributes

Certificate Selection

- Problem: I have certs signed with different algorithms
  - Need to somehow select one
  - New TLS extension: `cert_hash_types`

Per-record MAC

- Not much to worry about here
  - HMAC doesn't depend on collision-resistance
  - Not clear about how much preimage-resistance is required
- Already part of TLS negotiation
  - New cipher suites being created

KDF

- HMAC-based PRF construction
  - XOR SHA-1 and MD5 values
  - Belt-and-suspenders
    - ∗ Oops
- Switching to a negotiated PRF
  - Old PRF construction with stronger hashes
  - Alternate PRF constructions (GOST, NIST 800-56)
- This applies to Finished PRF too

IPsec

- Mostly the same issues as SSL/TLS
  - Which certificates to use if you have more than one?
  - Which hash functions initiator should use in "revised public key mode"
- Heuristics proposed in [Bell06]
- IETF doesn't plan any changes

Summary

- Existing systems aren't that threatened by current attacks
  - But enough to cause widespread protocol redesigns
- There's a surprising amount of robustness in our protocols
  - But certificates are a big single point of failure
- But future attacks could be extremely bad
- We're in a race with analytic progress