

# Deploying a New Hash Function

Steven M. Bellovin  
Columbia University  
smb@cs.columbia.edu

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-512
  - Only MD5 and SHA-1 are commonly used
  - No good theoretical foundation for constructions
- Attacks on MD5 and SHA-1 published in 2004-2005
  - Time to think about changing
- We studied the problem of transitioning to new functions
  - In TLS, S/MIME, IPsec
  - It's a mess

# 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.]

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

Important limitations:

- None of these attacks allows you to compute a preimage
- The colliders are not totally controllable
- Which pair collides depends on current hash state

# Practical Implications

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

- Not known how to extend this to different names
- Easy countermeasures available

**Non-repudiation** Daum and Lucks demonstrate a pair of Postscript documents with the same hash but different display output

- Digital signatures don't really work this way [Laurie]
- Easy to detect this attack

# Overview of the Transition Problem

- Types of implementations
  - Old** support only old algorithms
  - New** support only new algorithms
  - Switch-hitting** support both
- More complicated here because of certificates
  - A switch-hitting implementation can have either or both certificates

# 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
- In theory protocols already do this....

# Use of Hash Functions in S/MIME

- Message digital signatures
- Digital signatures in certificates
- Message MACs
- Key expansion (DH mode)

## The Initial Message: Signed Only

- Assume sender has two certificates
  - Otherwise there's no choice
- Why not sign twice (once with each certificate)?
  - S/MIME standard isn't totally clear here
  - Some implementations don't like partially verifiable messages
- Sender has no information about receiver's capabilities
  - Either certificate is potentially wrong
  - Have to guess based on overall upgrade rate
- Hash choice: use whatever is in the certificate

# The Initial Message: Encrypted

- Only a problem if also signed
  - Or DH mode is used (rarely)
- You have the recipient's certificate
  - Assume recipient can verify their own certificate
  - Use the same hash function as used there
- Potential fix: extension in certificate
  - RFC 4265 describes this
  - But it's not widely implemented yet

# Subsequent Messages

- If hash function capabilities are available
  - Use best common algorithm
  - Note: capabilities are signed
  - No need to send capabilities if already signing with best algorithm
- Else use whatever algorithm was previously used
  - You know that works
  - Though it may not be the best

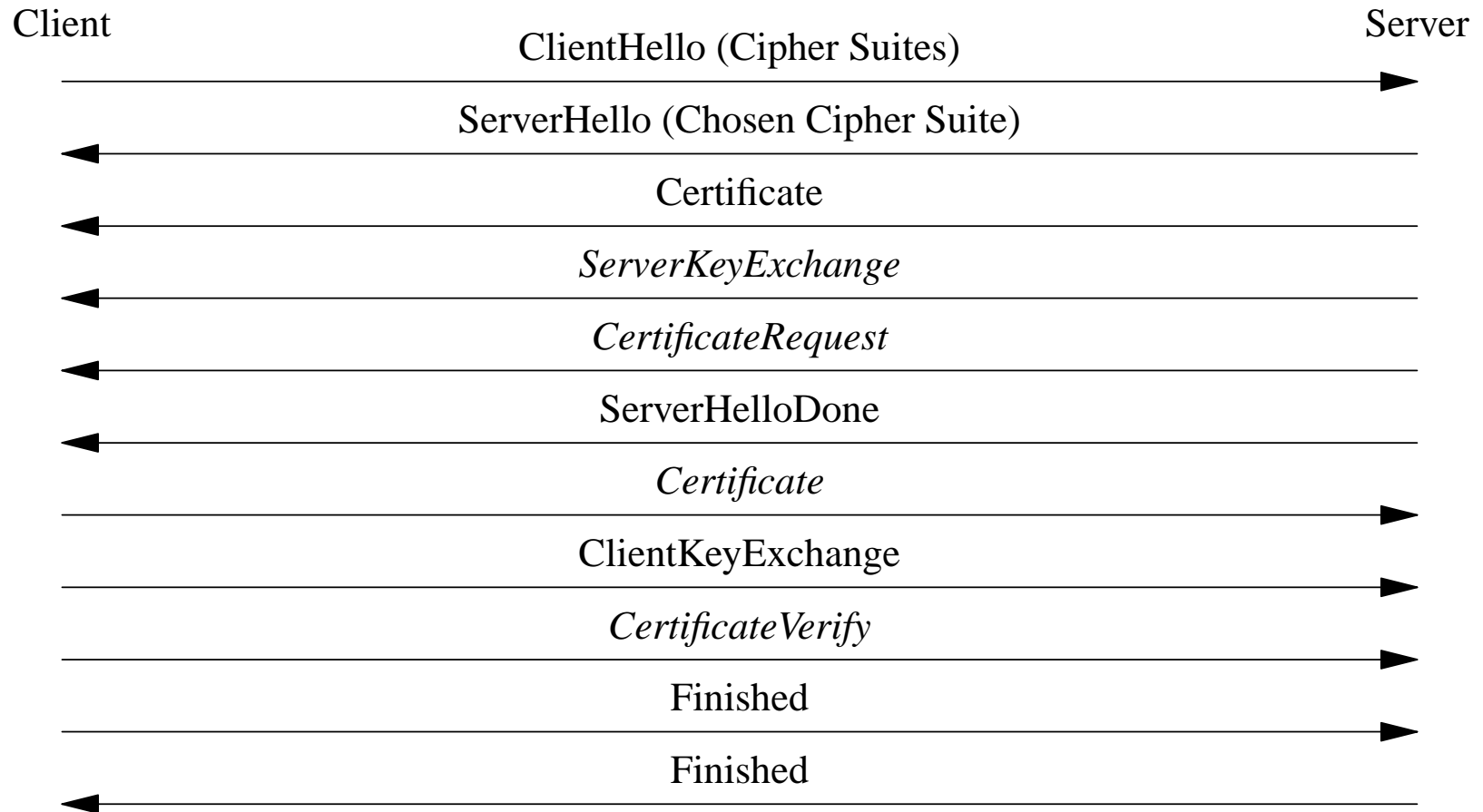
# Attacks

- Strip one signature
  - Not very useful unless you can compute a preimage
- Attack the certificate infrastructure
  - Lenstra-style
  - We don't know how to do this with current attacks
  - Easy to modify certificates to compensate
- Usual collision-based attacks
  - Sender pretends to only support a weak hash
    - \* Generates a collision with that hash
  - Standard defenses...

# Uses of Hash Functions in TLS

- Individually and negotiable
  - Per-record MAC
  - Certificates (sort of)
- MD5 and SHA-1 hardwired
  - Digitally-signed element (for ServerKeyExchange and CertificateVerify)
  - PRF
  - Finished message

# Review of TLS Negotiation



## 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
  - Just need to create new cipher suites

# Certificate Selection

- Server certificate
  - No negotiation for this at all presently
  - Options:
    - \* Bootstrap cipher suites
    - \* Use an extension (Best)
- Client certificate
  - Negotiated
    - \* But not for this
  - Options
    - \* Add extension
    - \* Expand current negotiation

# Digitally-signed

- RSA
  - Sign a concatenated MD5/SHA of handshake messages
- DSA/ECC
  - Sign a SHA-1 hash
- Idea: replace with a single hash function
  - Again, how to negotiate this?
  - Options:
    - \* Extension
    - \* Bootstrap off cipher suite
    - \* Whatever was in the certificate (Best)

# PRF

- Currently XOR of HMAC-SHA1 and HMAC-MD5
  - Replace with HMAC of negotiated cipher suite hash
- Wait: Finished messages provide downgrade protection
  - Only as strong as weakest common hash function
  - We're now in the business of approving/disapproving algorithms
    - \* Hard to get around this
    - \* Reminder: it's mostly preimages we care about
- Additional problem with Finished
  - Current structure:  $PRF(H(\textit{Handshake\_messages}))$
  - This avoids the need to buffer (key is first input to PRF)
  - Do we retain this structure?

# Other Protocols

- We also looked at IPsec, DNSSEC
- Housley has looked at OCSP
- All have problems
- IETF is currently working on transitions
  - But these are generic enhancements
  - Expect this to take 3-5 years

# Summary

- In principle our protocols are algorithm-agile
  - In practice, not so much
- This shouldn't surprise you
  - Nothing works until it's tested
- The fixes are straightforward
  - But annoyingly disruptive
- Expect it to take a long time