What’s the worst that could happen?

Eric Rescorla
RTFM, Inc.
DIMACS Workshop on Cryptography: Theory Meets Practice
Overview

- Cryptography alone doesn’t do much
  - Real systems combine primitives into protocols
- Protocols treat primitives as black boxes
  - With certain idealized properties
    - Indistinguishability, collision-freeness, preimage resistance...
  - The primitives only approximate those properties
    - Sometimes more than others...
- What happens when the primitives fail?
  - Let’s look at some plausible scenarios
Major cryptographic algorithms

- **Key establishment**
  - **RSA, DH**

- **Signature**
  - **RSA, DSS**

- **Encryption**
  - **DES, 3DES, AES, RC4, Blowfish**

- **Message digests**
  - **MD5, SHA-1, MD2**
Current status of key est. algorithms

**RSA**
- Basically sound but some active attacks
  - Million message attack
  - Timing analysis
- There are crypto countermeasures
  - OAEP, KEM, etc.
- In reality Countermeasures are implementation only
  - Both these attacks caused SSL implementation upgrades

**DH**
- Basically sound but some active attacks
  - Small subgroup
  - Timing analysis
- Again, implementation countermeasures
  - Most implementations use a fresh key for each transaction
Current status of signature algorithms

- **RSA**
  - Basically sound
  - Provable variants exist but aren’t used

- **DSS**
  - Believed to be basically sound
  - Limited by key length but NSA is extending
Current status of encryption algorithms (I)

**DES**
- Best analytic attacks require $2^{43}$ known plaintexts
  - In practice this has had no effect
- 56-bit key is known to be too weak
  - DES keys can be cracked in < 1 day for order $100k$ fixed cost

**3DES**
- No good analytic attacks
- Effective key strength ~112 bits
  - (3-key version)
Current status of encryption algorithms (II)

- **AES**
  - So far basically sound

- **RC4**
  - Some serious flaws
    - First 256-768 or so bytes are somewhat predictable [Mironov 02]
    - Related key vulnerabilities [Fluhrer and Shamir 01]
      - Structured keys are a real problem
  - Still widely used
Current status of digest algorithms

**MD5**
- Collisions are easy to find [Wang et al. 04]
  - ... however, they don’t appear to be controllable
    - Relationship between M and M’ is fixed
- Preimages are still difficult
- Still believed safe in HMAC

**SHA-1**
- So far appears sound
- Some disturbing results [Biham 04]
  - But only real progress is on reduced round versions

**SHA-XXX**
- Unknown, but some scary results [Hawkes et al. 04]
Attack 1: Controllable MD5 collisions

Current MD5 collisions are tightly constrained
- Only positions 4, 11, 41 are not fixed
  - And it’s not clear they can be set to chosen values
- But it seems reasonable to believe this attack can be extended

Attack description:
- Given any prefix P and desired values V and V’
- Create two suffixes S and S’ where
  - $H(P || V || S) = H(P || V’ || S’)$

For example
- $S || V = “Pay $10 <plus garbage>”$
- $S’ || V’ = “Pay $50 <plus other garbage>”$
Practical implications of MD5 collisions

- No real effect on most protocols
  - SSL, IPsec, SSH, etc. use MD5 in three ways
    - Key expansion
    - MACs
    - Signatures
  - Not affected by collisions
- Two important cases
  - Signed S/MIME messages
  - Certificates
Classic collision attack
- Attacker generates two variants
  - $M_1 = \text{“I will pay Eric $1.00/hr”}$ (a bargain)
  - $M_2 = \text{“I will pay Eric $1000/hr”}$ (a rip-off)
- Attacker gets victim to sign $M_1$
- Then claims victim signed $M_2$
  - And he has evidence to prove it
- This makes the most sense with contracts

Small problems
- Remember that random garbage?
  - Real contracts don’t have that
- Victim has both variants

Big problem
- This isn’t how contracts actually work
Victim has both variants

- Victim originally had “good” variant
- The attacker wants to enforce “bad” variant

**Question**
- Which one generated the good/bad pair?
- Each party points the finger

**But in a lot of situations it’s obvious**
- “Unsolicited” messages *must* have been generated by sender
  - Because finding pre-images is still hard
- Otherwise, sender must claim that receiver sent him a message he signed verbatim

**Why were you using MD5 anyway?**
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
Collisions and certificates

- Attack 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

```plaintext
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
}
```
Dealing with the random pads

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]
Attack 2: 1st preimages

- Preimages hard to find for “standard” hashes

**Attack description:**
- Given some hash value X
- Find a message M s.t. H(M) = X
- Assumption:
  - M is effectively random
  - ... not controllable by attacker

**For example**
- S/Key responses are iterated hashes H(H(H(H(H(seed))))))
  - Used in reverse order
- If I see one response I can predict the next one

**Most scenarios involve 2\textsuperscript{nd} preimages**
Attack 2 variant: partial 1\textsuperscript{st} preimage

- **Attacker sees:**
  - Digest value
  - Some of digest inputs
  - Common situations
    - Challenge/response
    - MACs for protocol data

- **Attacker wants to forge future values**
  - Using secret data
Trivial challenge/response protocol

Attacker wants to find Key
- Can use it to forge future responses
- If Key and Challenge are in same block, then chances that preimage will be useful are small
- Assume Key is padded to a block multiple
  - As in HMAC
Attacking partial 1\textsuperscript{st} preimages

Problem definition:
- Given M and hash compression function
- Find state \texttt{st} Compress(\texttt{State},M) = X
  - For all future values of \texttt{M},X

Not the same as a preimage
- Since we need a specific state
- ... in order to forge future messages
- This isn’t possible in general
  - Is it possible for ordinary hashes?
Preimage != State

Contrived hash function
- CBC-MAC variant with a fixed key
- Zero about half the CBC residue bits
  - $H_0 = 0$
  - $H_{n+1} = E((H_n \& \text{MD5}(M_{n+1})) \^ M_{n+1})$

Preimages are found by decrypting

Consider the two block case
- Decrypting $H_2$ gives $(H_1 \& \text{MD5}(M_2)) \^ M_2$
- Attacker can recover $H_1 \& \text{MD5}(M_2)$
- But any other challenge $(M_2)$ will zero different bits
  - So can’t forge new responses
  - Though each response leaks different bits...
What if you could forge MACs?

Does this break protocols?
- It depends...

Authenticate then encrypt (SSL/TLS)
- Block ciphers
  - Can’t re-insert the MAC
  - And wouldn’t match the data in any case
- Stream ciphers
  - Can reinsert MAC
  - ... but only if you know the plaintext

Encrypt than authenticate (IPsec)
- Easy to do an existential forgery
- Hard to do a controlled one unless plaintext is known

SSH is weird
- Authenticate then encrypt (but not the MAC)
- Can reinsert MAC
  - But it doesn’t match the data
Attack 3: 2nd preimages

- **Attack description:**
  - Given some message $M$
  - Find some message $M'$ s.t. $H(M) = H(M')$

- **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 choice
  - Validity of all certs with this digest is now questionable
  - 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
2\textsuperscript{nd} preimages and other protocols

- 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

- So, this is bad...
Attack 4: Weakness in initial RC4 bytes

RC4 initial bytes known to be imperfect
- Recommendation: discard first 256 bytes
- But most protocols don’t do this
  - SSL/TLS in particular

Attack description:
- Extension of Mironov and Fluhrer/Shamir work
- Recover key information from initial keystream
- Don’t need to recover key
  - Just predict other initial bytes...
Consequences of Attack 4

- Attacker can recover connection plaintext
- Credit cards over HTTPS are particularly weak
  - First 4 plaintext bytes known
  - Next 28-32 (TLS) or 52-56 (SSLv3) plaintext bytes are random
  - Next plaintext bytes are HTTP fetch and header
    - 100-500 bytes
    - Very predictable
  - Followed by a credit card #
    - Predictable structure helps here
Countermeasures for Attack 4

In principle easy
- At least for SSL
  - 802.11 already moving to AES
- Almost all clients and servers support DES, 3DES, etc.
  - It’s a negotiable item
  - Server admin can just turn off RC4

In practice not so easy
- Admins are concerned about performance
- Uptake of fixes is very slow [Rescorla 03]

May not be the easiest attack
- You only recover 1 credit card number
- Poorly maintained servers may have other flaws
Attack 5: DES-quality attacks on AES/3DES

Current AES/3DES attacks are nearly useless
- What if we had attacks on AES as good as those on DES?

Attack description:
- Recover key with $2^{43}$ known plaintexts and $2^{43}$ ops
- This would be a major success
  - $2^{69}$ improvement for 3DES
  - $2^{85}$ improvement for AES

But what does it mean for a real system?
Implications for common protocols

- **SSL**
  - Each connection uses a separate key
  - Most connections are short (HTTP)
    - 5 minutes is considered long
- **SSH**
  - Longer but not a lot of data is moved
- **S/MIME**
  - Each message uses a separate key
  - When would you have part of a message in the clear?
  - $2^{43}$ blocks = $10^{14}$ bytes
    - This is longer than any commercial disk
    - So not realistic as a message
- **IPsec**
  - $2^{43}$ blocks is 10 days of full-speed 1Gig traffic
    - Not a common situation
  - This attack doesn’t apply to 3DES
    - 3DES uses CBC mode
    - You need to change keys every $2^{32}$ blocks anyway
Attack 5 Variant: Total cipher break

- Complete key recovery
  - Using a few known plaintexts
  - And relatively fast

- Compromises confidentiality

- No effect on authentication
  - Encryption keys decoupled from MAC keys
    - At least in well designed protocols
  - Often encryption keys too short to recover master secret
    - Even if PRFs were broken
Attack 6: Remote key recovery

- E.g., timing attacks [Kocher], [Boneh and Brumley 03]
  - Not known if can be executed over Internet
  - Easily fixed (blinding)

- Attack description:
  - Repeated remote probes allow recovery of private key
Implications of Attack 6

SSH, IPsec typically use DH
- With a fresh key for each exchange
- Attacks on signature?
  - No control of plaintext
- Can’t attack connection A from connection B
- ... SSHv1 was weaker...

SSL/TLS
- Generally uses static RSA
  - Though DH variants exist
- These attacks work well here

S/MIME
- What about automated mail responders?
  - Timing?
  - Faults?
Attack 7: RSA signature malleability

- Signature forgery is obviously a disaster
  - What about something weaker?

- Attack description:
  - Given a signature over message M
    - actually hash value M
    - modify the last few bits

- Not very plausible with RSA
  - PKCS-1 padding
  - What about DSA?

- But not message integrity
  - Can’t go from encryption keys to MAC keys
    - Both are generated from a master key
  - Even broken hashes don’t help
    - Master keys are too long
Implications of signature malleability

Remember: all signatures are over hashes
- Forged signature is over a random value
  - Effectively an existential forgery
  - Note: many algorithms already have this property
- Need to find usable preimage

Use a meet-in-the-middle attack
- $2^{n/2}$ operations
- $2^{n/2}$ storage
- Can’t be done in real time....

Only practical for very high value transactions
- Unless of course the hash was also broken
Take home points

- Protocols are surprisingly resistant failure to primitive
- Randomness really helps
- Timing counts
- Hash early, hash often
- Sometimes it’s better to be lucky than good
Major comsec protocols

- **SSL/TLS**: Application layer generic channel security
  - Web traffic
  - E-mail (SMTP/TLS)
  - SSL VPNs...
  - Mostly short-lived connections between client and server
- **SSH**: Application layer channel security
  - Remote login
- **IPsec**: Network-level channel security
  - VPNs
  - Long-term associations between networks
- **S/MIME, PGP**: Application layer message security
  - E-mail