Computer Security and Privacy

Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Goals for Today

🔹 Cryptography (Continued)
  • Symmetric cryptography

🔹 Research:
  • Self-destructing data
First

- Under the hood: Symmetric encryption
Achieving Privacy (Symmetric)

Encryption schemes: A tool for protecting privacy.

Message \[ \rightarrow \text{Encrypt} \rightarrow \text{Decrypt} \rightarrow \text{Ciphertext} \rightarrow \text{Ciphertext} \rightarrow \text{Message} \]

Alice \[ K \]

Bob \[ K \]

Adversary
Attack Scenarios for Encryption

- Ciphertext-Only
- Known Plaintext
- Chosen Plaintext
- Chosen Ciphertext (and Chosen Plaintext)

(General advice: Target strongest level of privacy possible -- even if not clear why -- for extra “safety”)

Chosen-Plaintext Attack

Crook #1 changes his PIN to a number of his choice

PIN is encrypted and transmitted to bank

Crook #2 eavesdrops on the wire and learns ciphertext corresponding to chosen plaintext PIN

... repeat for any PIN value
Achieving Privacy (Symmetric)

Encryption schemes: A tool for protecting privacy.

Message .......... M
Ciphertext .......... C

Adversary

Alice
K

Bob
K
Achieving Integrity (Symmetric)

Message authentication schemes: A tool for protecting integrity.
(Also called message authentication codes or MACs.)

Message authentication schemes: A tool for protecting integrity.
(Also called message authentication codes or MACs.)
Attack Scenarios for Integrity

What do you think these scenarios should be?
Achieving Integrity (Symmetric)

Message authentication schemes: A tool for protecting integrity.
(Also called message authentication codes or MACs.)
One-Time Pad

Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
\[ \text{ciphertext} = \text{plaintext} \oplus \text{key} \]

Decrypt by bitwise XOR of ciphertext and key:
\[ \text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \oplus (\text{key} \oplus \text{key}) = \text{plaintext} \]

Cipher achieves **perfect secrecy** if and only if there are as many possible keys as possible plaintexts, and every key is equally likely  (Claude Shannon)
Advantages of One-Time Pad

◆ Easy to compute
  • Encryption and decryption are the same operation
  • Bitwise XOR is very cheap to compute
◆ As secure as theoretically possible
  • Given a ciphertext, all plaintexts are equally likely, regardless of attacker’s computational resources
  • ...as long as the key sequence is truly random
    – True randomness is expensive to obtain in large quantities
  • ...as long as each key is same length as plaintext
    – But how does the sender communicate the key to receiver?
Disadvantages

Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
\[ \text{ciphertext} = \text{plaintext} \oplus \text{key} \]

Decrypt by bitwise XOR of ciphertext and key:
\[ \text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \]

Disadvantage #1: Keys as long as messages.
Impractical in most scenarios
(Supposedly still used by intelligence communities)
Disadvantages

Key is a random bit sequence as long as the plaintext

Encrypt by bitwise XOR of plaintext and key:
\[ \text{ciphertext} = \text{plaintext} \oplus \text{key} \]

Decrypt by bitwise XOR of ciphertext and key:
\[ \text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \]

Disadvantage #2: No integrity protection
Disadvantages

Disadvantage #3: Keys cannot be reused

Learn relationship between plaintexts:
\[ C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = (P_1 \oplus P_2) \oplus (K \oplus K) = P_1 \oplus P_2 \]
Reducing Keysize

- What do we do when we can’t pre-share huge keys?
  - When OTP is unrealistic

- We use special cryptographic primitives
  - Single key can be reused (with some restrictions)
  - But no longer provable secure (in the sense of the OTP)

- Examples: Block ciphers, stream ciphers
Background: Permutation

- For N values, N! possible permutations
- For N-bit inputs, $2^N!$ possible permutations!
- Idea for how to use: split plaintext into blocks; for each block use secret key to pick a permutation
  - Without the key, permutation should “look random”
Block Ciphers

- Operates on a single chunk ("block") of plaintext
  - For example, 64 bits for DES, 128 bits for AES
  - Same key is reused for each block (can use short keys)
Block Cipher Security

- Result should look like a random permutation
  - “As if” plaintext bits were randomly shuffled

- Only computational guarantee of secrecy
  - Not impossible to break, just very expensive
    - If there is no efficient algorithm (unproven assumption!), then can only break by brute-force, try-every-possible-key search
  - Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information
Block Cipher Operation (Simplified)

- Block of plaintext
  - Add some secret key bits to provide confusion
  - Each S-box transforms its input bits in a "random-looking" way to provide diffusion (spread plaintext bits throughout ciphertext)
  - Procedure must be reversible (for decryption)
- Block of ciphertext
  - repeat for several rounds

Key
Feistel Structure (Stallings Fig 2.2)
DES

◆ Feistel structure
  • “Ladder” structure: split input in half, put one half through the round and XOR with the other half
  • After 3 random rounds, ciphertext indistinguishable from a random permutation if internal F function is a pseudorandom function (Luby & Rackoff)
    – Theoretical results: Evidence that the “ladder” structure is a solid design
    – In practice: Use more than 3 rounds

◆ DES: Data Encryption Standard
  • Feistel structure
  • Invented by IBM, issued as federal standard in 1977
  • 64-bit blocks, 56-bit key + 8 bits for parity
## DES and 56 bit keys (Stallings Tab 2.2)

- **56 bit keys are quite short**

<table>
<thead>
<tr>
<th>Key Size (bits)</th>
<th>Number of Alternative Keys</th>
<th>Time required at 1 encryption/$\mu$s</th>
<th>Time required at $10^6$ encryptions/$\mu$s</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>$2^{32} = 4.3 \times 10^9$</td>
<td>$2^{31} \mu s = 35.8$ minutes</td>
<td>2.15 milliseconds</td>
</tr>
<tr>
<td>56</td>
<td>$2^{56} = 7.2 \times 10^{16}$</td>
<td>$2^{55} \mu s = 1142$ years</td>
<td>10.01 hours</td>
</tr>
<tr>
<td>128</td>
<td>$2^{128} = 3.4 \times 10^{38}$</td>
<td>$2^{127} \mu s = 5.4 \times 10^{24}$ years</td>
<td>$5.4 \times 10^{18}$ years</td>
</tr>
<tr>
<td>168</td>
<td>$2^{168} = 3.7 \times 10^{50}$</td>
<td>$2^{167} \mu s = 5.9 \times 10^{36}$ years</td>
<td>$5.9 \times 10^{30}$ years</td>
</tr>
<tr>
<td>26 characters (permutation)</td>
<td>$26! = 4 \times 10^{26}$</td>
<td>$2 \times 10^{26} \mu s = 6.4 \times 10^{12}$ years</td>
<td>$6.4 \times 10^{6}$ years</td>
</tr>
</tbody>
</table>

- **1999: EFF DES Crack + distributed machines**
  - < 24 hours to find DES key
- **DES ---> 3DES**
  - 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)
Advanced Encryption Standard (AES)

- New federal standard as of 2001
- Based on the Rijndael algorithm
- 128-bit blocks, keys can be 128, 192 or 256 bits
- Unlike DES, does not use Feistel structure
  - The entire block is processed during each round
- Design uses some very nice mathematics
Basic Structure of Rijndael

128-bit plaintext
(arranged as 4x4 array of 8-bit bytes)

⊕

S

byte substitution

Shift rows
(1\textsuperscript{st} unchanged, 2\textsuperscript{nd} left by 1, 3\textsuperscript{rd} left by 2, 4\textsuperscript{th} left by 3)

Mix columns
(each new byte depends on all bytes in old column)

⊕

128-bit key

Repeat 10 times

add key for this round

Expand key
Encrypting a Large Message

So, we’ve got a good block cipher, but our plaintext is larger than 128-bit block size

Electronic Code Book (ECB) mode

- Split plaintext into blocks, encrypt each one separately using the block cipher

Cipher Block Chaining (CBC) mode

- Split plaintext into blocks, XOR each block with the result of encrypting previous blocks

Counter (CTR) mode

- Use block cipher to generate keystream, like a stream cipher

...
ECB Mode

- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks
CBC Mode: Encryption

- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity
CBC Mode: Decryption

plaintext

Initialization vector

Key

decrypt

ciphertext
ECB vs. CBC

AES in ECB mode

Similar plaintext blocks produce similar ciphertext blocks (not good!)

AES in CBC mode
Information Leakage in ECB Mode

Encrypt in ECB mode
CBC and Electronic Voting

Initialization vector (supposed to be random)

plaintext $\oplus$ Key $\rightarrow$ DES $\rightarrow$ ciphertext

Found in the source code for Diebold voting machines:

```
DesCBCEncrypt((des_c_block*)tmp, (des_c_block*)record.m_Data, 
              totalSize, DESKEY, NULL, DES_ENCRYPT)
```
CTR Mode: Encryption

- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity
- Fragile if ctr repeats
CTR Mode: Decryption

Initial ctr

ctr → Key → block cipher

ctr+1 → Key → block cipher

ctr+2 → Key → block cipher

ctr+3 → Key → block cipher

ct ⊕ pt

Key

block cipher

pt
Next

- Defining security for symmetric encryption
When Is a Cipher “Secure”?  

- **Hard to recover the key?**  
  - What if attacker can learn plaintext without learning the key?  

- **Hard to recover plaintext from ciphertext?**  
  - What if attacker learns some bits or some function of bits?  

- **Fixed mapping from plaintexts to ciphertexts?**  
  - What if attacker sees two identical ciphertexts and infers that the corresponding plaintexts are identical?  
  - *(Implication: encryption must be randomized or stateful)*
How Can a Cipher Be Attacked?

- Assume that the attacker knows the encryption algorithm and wants to decrypt some ciphertext
- Main question: what else does attacker know?
  - Depends on the application in which cipher is used!
- Ciphertext-only attack
- Known-plaintext attack (stronger)
  - Knows some plaintext-ciphertext pairs
- Chosen-plaintext attack (even stronger)
  - Can obtain ciphertext for any plaintext of his choice
- Chosen-ciphertext attack (very strong)
  - Can decrypt any ciphertext except the target
  - Sometimes very realistic model
Defining Security (Not Required)

- Attacker does *not know* the key
- He chooses as many plaintexts as he wants, and learns the corresponding ciphertexts
- When ready, he picks two plaintexts $M_0$ and $M_1$
  - He is even allowed to pick plaintexts for which he previously learned ciphertexts!
- He receives either a ciphertext of $M_0$, or a ciphertext of $M_1$
- He wins if he guesses correctly which one it is
Defining Security (Not Required)

- Idea: attacker should not be able to learn even a single bit of the encrypted plaintext
- Define Enc($M_0$, $M_1$, $b$) to be a function that returns encrypted $M_b$
  - Given two plaintexts, Enc returns a ciphertext of one or the other depending on the value of bit $b$
  - Think of Enc as a magic box that computes ciphertexts on attacker’s demand. He can obtain a ciphertext of any plaintext $M$ by submitting $M_0=M_1=M$, or he can try to learn even more by submitting $M_0 \neq M_1$.
- Attacker’s goal is to learn just one bit $b$
Chosen-Plaintext Security (Not Required)

Consider two experiments (A is the attacker)

Experiment 0

A interacts with Enc(-,-,0)
and outputs bit d

• Identical except for the value of the secret bit

• d is attacker’s guess of the secret bit

Experiment 1

A interacts with Enc(-,-,1)
and outputs bit d

Attacker’s advantage is defined as

\[ | \text{Prob}(A \text{ outputs 1 in Exp0}) - \text{Prob}(A \text{ outputs 1 in Exp1})) | \]

Encryption scheme is chosen-plaintext secure if this advantage is negligible for any efficient A
“Simple” Example (Not Required)

❖ Any deterministic, stateless symmetric encryption scheme is insecure
  • Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
  • This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

Let X, Y be any two different plaintexts

C₁ ← Enc(X, Y, b);   C₂ ← Enc(Y, Y, b);

If C₁ = C₂ then b = 1 else say b = 0

❖ The advantage of this attacker A is 1

Prob(A outputs 1 if b = 0) = 0   Prob(A outputs 1 if b = 1) = 1
Why Hide Everything?

- Leaking even a little bit of information about the plaintext can be disastrous
- Electronic voting
  - 2 candidates on the ballot (1 bit to encode the vote)
  - If ciphertext leaks the parity bit of the encrypted plaintext, eavesdropper learns the entire vote
- Also, want a strong definition that implies others
Birthday attacks

Are there two people in this classroom that have the same birthday?

• Yes?
• No?
Birthday attacks

Why is this important for cryptography?

- 365 days in a year (366 some years)
  - Pick one person. To find another person with same birthday would take on the order of 365/2 = 182.5 people
  - Expect “collision” -- two people with same birthday -- with a room of only 23 people
  - For simplicity, approximate when we expect a collision as the square root of 365.

- $2^{128}$ different 128-bit keys
  - Pick one key at random. To exhaustively search for this key requires trying on average $2^{127}$ keys.
  - Expect a “collision” after selecting approximately $2^{64}$ random keys.
  - 64 bits of security against collision attacks, not 128 bits.
Next

- Under the hood: Hash functions and MACs
Integrity

Software manufacturer wants to ensure that the executable file is received by users without modification. It sends out the file to users and publishes its hash in NY Times. The goal is **integrity**, not secrecy.

Idea: given goodFile and hash(goodFile), very hard to find badFile such that hash(goodFile)=hash(badFile)
Integrity vs. Secrecy

- **Integrity**: attacker cannot tamper with message
- **Encryption does not always guarantee integrity**
  - Intuition: attacker may able to modify message under encryption without learning what it is
    - One-time pad: given key K, encrypt M as $M \oplus K$
    - This guarantees perfect secrecy, but attacker can easily change unknown M under encryption to $M \oplus M'$ for any $M'$
    - Online auction: halve competitor’s bid without learning its value
  - This is recognized by industry standards (e.g., PKCS)
    - “RSA encryption is intended primarily to provide confidentiality... It is not intended to provide integrity” (from RSA Labs Bulletin)
Hash Functions: Main Idea

- H is a lossy compression function
  - Collisions: $h(x) = h(x')$ for distinct inputs $x, x'$
  - Result of hashing should "look random" (make this precise later)
    - Intuition: half of digest bits are "1"; any bit in digest is "1" half the time

- Cryptographic hash function needs a few properties...
One-Way

- **Intuition:** hash should be hard to invert
  - “Preimage resistance”
  - Let $h(x') = y \in \{0, 1\}^n$ for a random $x'$
  - Given $y$, it should be hard to find any $x$ such that $h(x) = y$

- **How hard?**
  - Brute-force: try every possible $x$, see if $h(x) = y$
  - SHA-1 (common hash function) has 160-bit output
    - Expect to try $2^{159}$ inputs before finding one that hashes to $y$. 

Collision Resistance

- Should be hard to find distinct $x, x'$ such that $h(x) = h(x')$
  - Brute-force collision search is only $O(2^{n/2})$, not $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$

- Birthday paradox (informal)
  - Let $t$ be the number of values $x, x', x''...$ we need to look at before finding the first pair $x, x'$ s.t. $h(x) = h(x')$
  - What is probability of collision for each pair $x, x'$? $1/2^n$
  - How many pairs would we need to look at before finding the first collision? $O(2^n)$
  - How many pairs $x, x'$ total? $\text{Choose}(t, 2) = t(t-1)/2 \sim O(t^2)$
  - What is $t$? $2^{n/2}$
One-Way vs. Collision Resistance

- **One-wayness does not imply collision resistance**
  - Suppose \( g \) is one-way
  - Define \( h(x) = g(x') \) where \( x' \) is \( x \) except the last bit
    - \( h \) is one-way (to invert \( h \), must invert \( g \))
    - Collisions for \( h \) are easy to find: for any \( x \), \( h(x_0) = h(x_1) \)

- **Collision resistance does not imply one-wayness**
  - Suppose \( g \) is collision-resistant
  - Define \( h(x) \) to be \( 0x \) if \( x \) is \( n \)-bit long, \( 1g(x) \) otherwise
    - Collisions for \( h \) are hard to find: if \( y \) starts with 0, then there are no collisions, if \( y \) starts with 1, then must find collisions in \( g \)
    - \( h \) is not one way: half of all \( y \)'s (those whose first bit is 0) are easy to invert (how?); random \( y \) is invertible with probab. 1/2
Weak Collision Resistance

- Given randomly chosen $x$, hard to find $x'$ such that $h(x) = h(x')$
  - Attacker must find collision for a specific $x$. By contrast, to break collision resistance, enough to find any collision.
  - Brute-force attack requires $O(2^n)$ time
  - AKA second-preimage collision resistance
- Weak collision resistance does not imply collision resistance
Which Property Do We Need?

- UNIX passwords stored as hash(password)
  - One-wayness: hard to recover password
  - Second-preimage resistance: hard to recover “equivalent” passwd

- Integrity of software distribution
  - Weak collision resistance
  - But software images are not really random... maybe need full collision resistance

- Auction bidding
  - Alice wants to bid B, sends H(B), later reveals B
  - One-wayness: rival bidders should not recover B
  - Collision resistance: Alice should not be able to change her mind to bid B’ such that H(B)=H(B’)

Common Hash Functions

- **MD5**
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)

- **RIPEMD-160**
  - 160-bit variant of MD5

- **SHA-1 (Secure Hash Algorithm)**
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Also recently broken! (Theoretically -- not practical.)

- **SHA-256, SHA-512, SHA-224, SHA-384**
- **SHA-3**: Forthcoming.
Basic Structure of SHA-1 (Not Required)

- **Split message into 512-bit blocks**

- **Compression function**
  - Applied to each 512-bit block and current 160-bit buffer
  - This is the heart of SHA-1

- **Against padding attacks**

- **Message length (K mod 2^64)**

- **Padding (1 to 512 bits)**

- **L x 512 bits = N x 32 bits**

- **160-bit buffer (5 registers)**
  - Initialized with magic values
How Strong Is SHA-1?

- Every bit of output depends on every bit of input
  - Very important property for collision-resistance
- Brute-force inversion requires $O(2^{160})$ ops, birthday attack on collision resistance requires $O(2^{80})$ ops
- Some very recent weaknesses (2005)
  - Collisions can be found in $2^{63}$ ops
Common Hash Functions

◆ MD5
  • 128-bit output
  • Designed by Ron Rivest, used very widely
  • Collision-resistance broken (summer of 2004)

◆ RIPEMD-160
  • 160-bit variant of MD5

◆ SHA-1 (Secure Hash Algorithm)
  • 160-bit output
  • US government (NIST) standard as of 1993-95
  • Also recently broken! (Theoretically -- not practical.)

◆ SHA-256, SHA-512, SHA-224, SHA-384

◆ SHA-3: Forthcoming.
International Criminal Tribunal for Rwanda (Which Properties of Hash Functions?)


- Credits: Alexei Czeskis, Karl Koscher, Batya Friedman
Achieving Integrity (Symmetric)

Message authentication schemes: A tool for protecting integrity.

(Also called message authentication codes or MACs.)
HMAC

- Construct MAC by applying a cryptographic hash function to message and key

- Original motivation:
  - Could also use encryption instead of hashing, but...
  - Hashing is faster than encryption in software
  - Library code for hash functions widely available
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption

- Invented by Bellare, Canetti, and Krawczyk (1996)
  - HMAC strength established by cryptographic analysis

- Mandatory for IP security, also used in SSL/TLS
Structure of HMAC

- Secret key padded to block size
- Magic value (flips half of key bits)
- Block size of embedded hash function
- Another magic value (flips different key bits)
- Embedded hash function (strength of HMAC relies on strength of this hash function)
- "Black box": can use this HMAC construction with any hash function (why is this important?)
- "Amplify" key material (get two keys out of one)
- Very common problem: given a small secret, how to derive a lot of new keys?
- hash(key, hash(key, message))
CBC Mode: Encryption

- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity
CBC-MAC

- Not secure when system may MAC messages of different lengths.
  - Encode length at beginning
  - Use a derivative called CMAC
- Internal collisions and birthday attacks
Example attacks to CBC-MAC

Example problems (for whiteboard):
- When process messages of different length
- When process messages of different length and encode length at end of message
- When a collision occurs

Recommendations
- Include length at beginning of message before CBC-MAC
- Use CMAC (which is like CBC-MAC but handles last block differently)
Achieving Both Privacy and Integrity

Authenticated encryption scheme

Recall: Often desire both privacy and integrity. (For SSH, SSL, IPsec, etc.)
Some subtleties! Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.
But insecure! [BN, Kra]

Assume Alice sends messages:

If $T_i = T_j$ then $M_i = M_j$

Adversary learns whether two plaintexts are equal.

Especially problematic when $M_1, M_2, \ldots$ take on only a small number of possible values.
Results of [BN00, Kra01]

<table>
<thead>
<tr>
<th>Privacy</th>
<th>Integrity</th>
<th>Encryption Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong (CCA)</td>
<td>Strong (CTXT)</td>
<td>Encrypt-then-MAC</td>
</tr>
<tr>
<td>Weak (CPA)</td>
<td>Weak (PTXT)</td>
<td>MAC-then-Encrypt</td>
</tr>
<tr>
<td>Insecure</td>
<td>Weak (PTXT)</td>
<td>Encrypt-and-MAC</td>
</tr>
</tbody>
</table>

- **Privacy**
  - Strong (CCA)
  - Weak (CPA)
  - Insecure

- **Integrity**
  - Strong (CTXT)
  - Weak (PTXT)
  - Weak (PTXT)
The Secure Shell (SSH) protocol is designed to provide:

- Secure remote logins.
- Secure file transfers.

Where security includes:

- Protecting the privacy of users’ data.
- Protecting the integrity of users’ data.

OpenSSH is included in the default installations of OS X and many Linux distributions.
Authenticated encryption in SSH

\[ \overline{E^{K_e, K_m}} \]

Data to be communicated

Maintained internally; not transmitted

\[ \begin{array}{c}
\text{ctr} \\
\text{pl} \\
pdl \\
M \\
padding
\end{array} \]

4 bytes
4 bytes
1 byte

\[ \text{Encrypt}_{K_e} \]

\[ \text{MAC}_{K_m} \]

\[ C' \]

\[ T \]

Ciphertext packet
Assume Alice sends messages \( M_1 \) and \( M_2 \) that are the same.

What's different about SSH?

Then the tags \( T_1 \) and \( T_2 \) will be different with high probability.

But if counters repeat, tags may once again leak private information about data.
Next

- A bit more context...
RFIDs in car keys:

• RFIDs in car keys make it harder to hotwire a car
• Result: Car jackings increased

Improved security, increased risk

Biometric car lock defeated by cutting off owner's finger

POSTED BY CORY DOCTOROW, MARCH 31, 2005 7:53 AM |
PERMALINK

Andrei sez, "'Malaysia car thieves steal finger.' This is what security visionaries Bruce Schneier and Ross Anderson have been warning about for a long time. Protect your $75,000 Mercedes with biometrics and you risk losing whatever body part is required by the biometric mechanism."

"...[H]aving stripped the car, the thieves became frustrated when they wanted to restart it. They found they again could not bypass the immobiliser, which needs the owner's fingerprint to disarm it.

They stripped Mr Kumaran naked and left him by the side of the road - but not before cutting off the end of his index finger with a machete."