Algorithms and Mechanisms

Cryptography is nothing more than a mathematical framework for discussing the implications of various paranoid delusions

— Don Alvarez

Historical Ciphers

Non-standard hieroglyphics, 1900BC

Atbash cipher (Old Testament, reversed Hebrew alphabet, 600BC)

Caesar cipher:

letter = letter + 3

‘fish’ → ‘ilvk’

rot13: Add 13/swap alphabet halves

- Usenet convention used to hide possibly offensive jokes
- Applying it twice restores the original text

Substitution Ciphers

Simple substitution cipher:

a = p, b = m, c = f, ...

- Break via letter frequency analysis

Polyalphabetic substitution cipher

1. a = p, b = m, c = f, ...
2. a = l, b = t, c = a, ...
3. a = f, b = x, c = p, ...

- Break by decomposing into individual alphabets, then solve as simple substitution

One-time Pad (1917)

Message	s	e	c	r	e	t
	18	5	3	17	5	19
OTP	+15	8	1	12	19	5
	7	13	4	3	24	24
	g	m	d	c	x	x

OTP is unbreakable *provided*

- Pad is never reused (VENONA)
- Unpredictable random numbers are used (physical sources, e.g. radioactive decay)

One-time Pad (ctd)

Used by

- Russian spies
- The Washington-Moscow “hot line”
- CIA covert operations

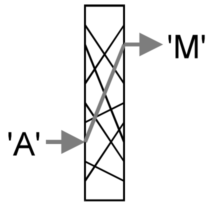
Many snake oil algorithms claim unbreakability by claiming to be a OTP

- Pseudo-OTPs give pseudo-security

Cipher machines attempted to create approximations to OTPs, first mechanically, then electronically

Cipher Machines (~1920)

1. Basic component = wired rotor



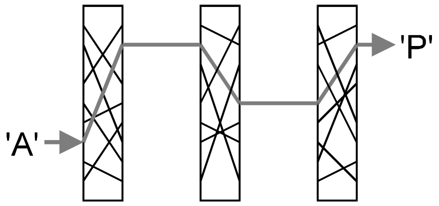
- Simple substitution

2. Step the rotor after each letter

- Polyalphabetic substitution, period = 26

Cipher Machines (ctd)

3. Chain multiple rotors



Each rotor steps the next one when a full turn is complete

Cipher Machines (ctd)

Two rotors, period = 26×26

= 676

Three rotors, period = $26 \times 26 \times 26$

= 17,576

Rotor sizes are chosen to be relatively prime to give maximum-length sequence

Key = rotor wiring, rotor start position

Cipher Machines (ctd)

Famous rotor machines

- US: Converter M-209
- UK: TYPEX
- Japan: Red, Purple
- Germany: Enigma

Many books on Enigma

- Kahn, Seizing the Enigma
- Levin, Ultra Goes to War
- Welchman, The Hut Six Story
- Winterbotham, The Ultra Secret

"It would have been secure if used properly"

Use of predictable openings:

- "Mein Fuehrer! ..."
- "Nothing to report"

Use of the same key over an extended period

Encryption of the same message with old (compromised) and new keys

- Post-war KW-26 common fill device shredded the key card when the cover was opened to prevent this

Device treated as a magic black box, a mistake still made today

Inventors believed it was infallible, " " " " "

Cipher Machines (ctd)

Various kludges were made to try to improve security — none worked

Enigmas were sold to friendly nations after the war

Improved rotor machines were used into the 70's and 80's

Further reading:

- Kahn, The Codebreakers
- Cryptologia, quarterly journal

Stream Ciphers

Binary pad (keystream), use XOR instead of addition

Plaintext = original, unencrypted data
Ciphertext = encrypted data

Plaintext		1	0	0	1	0	1	1
Keystream	XOR	0	1	0	1	1	0	1
Ciphertext		1	1	0	0	1	1	0
Keystream	XOR	0	1	0	1	1	0	1
Plaintext		1	0	0	1	0	1	1

Two XORs with the same data always cancel out

Stream Ciphers (ctd)

Using the keystream and ciphertext, we can recover the plaintext

but

Using the plaintext and ciphertext, we can recover the keystream

Using two ciphertexts from the same keystream, we can recover the XOR of the plaintexts

- Any two components of an XOR-based encryption will recover the third
- Never reuse a key with a stream cipher
- Better still, never use a stream cipher

Stream Ciphers (ctd)

Vulnerable to bit-flipping attacks

```
 Plaintext QT-TRNSFER USD 000010,00 FRM ACCNT 12345-67 TO
Ciphertext sSJNsF7BQIPBCjTUo1yl06VohNJcsALNpqf05xe9X0nYLd
                         00101101
                                ↓ Flip low bit
                         00101100
Ciphertext sSJNsF7BQIPBCjTTo1yl06VohNJcsALNpqf05xe9X0nYLd
 Plaintext QT-TRNSFER USD 100010,00 FRM ACCNT 12345-67 TO
```

RC4

Stream cipher optimised for fast software implementation

- 2048-bit key, 8-bit output

Formerly a trade secret of RSADSI

- Reverse-engineered and posted to the net in 1994

```
while( length-- )
  {
  x++; sx = state[ x ]; y += sx;
  sy = state[ y ]; state[ y ] = sx; state[ x ] = sy;
  *data++ ^= state[ ( sx+sy ) & 0xFF ];
  }
```

Takes about a minute to implement from memory

Extremely fast

RC4 (ctd)

Used in SSL (Netscape, MSIE), Lotus Notes, Windows password encryption, MS Access, Adobe Acrobat, MS PPTP, Oracle Secure SQL, ...

- Usually used in a manner that allows the keystream to be recovered (Windows password encryption, Windows server authentication, Windows NT SYSKEY, early Netscape server key encryption, some MS server/browser key encryption, MS PPTP, MS Access, MS Word, XBox, ...)
- *Every* MS product which is known to use it has got it wrong at some time over more than a decade (!!)

Illustrates the problem of treating a cipher as a magic black box

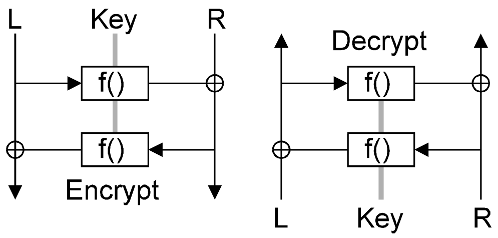
Recommendation: Avoid this, it’s too easy to get wrong

Block Ciphers

Originated with early 1970's IBM effort to develop banking security systems

First result was Lucifer, most common variant has 128-bit key and block size

- It wasn't secure in any of its variants



Called a Feistel or product cipher

Block Ciphers (ctd)

Key is applied via the f()-function

- A simple transformation
- Doesn't have to be reversible

Each step is called a round

- The more rounds, the greater the security (to a point)

Most famous example of this design is DES

- 16 rounds
- 56 bit key
- 64 bit block size (L,R = 32 bits)

Designed by IBM with advice from the NSA

Attacking Feistel Ciphers

Differential cryptanalysis

- Looks for correlations in f()-function input and output

Linear cryptanalysis

- Looks for correlations between key and cipher input and output

Related-key cryptanalysis

- Looks for correlations between key changes and cipher input/output

Differential cryptanalysis was (re-)discovered in 1990; virtually all block ciphers from before that time are vulnerable...

...except DES. IBM (and the NSA) knew about it 15 years earlier

Strength of DES

Key size = 56 bits

Brute force = 2^{55} attempts

Differential cryptanalysis = 2^{47} attempts

Linear cryptanalysis = 2^{43} attempts

- (but the last two are impractical)
- This type of attack is known as a certificational weakness

> 56 bit keys don't make it any stronger

- NSA didn't really weaken DES by setting the key size at 56 bits

> 16 rounds don't make it any stronger

DES Key Problems

Key size = 56 bits

= 8 × 7-bit ASCII chars

Alphanumeric-only password converted to uppercase

= 8 × ~5-bit chars

= 40 bits

DES uses the low bit in each byte for parity

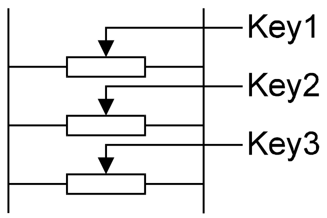
= 32 bits

- Many 1980s/early-90s DES programs used this form of keying
- Forgetting about the parity bits is so common that the NSA probably designs its keysearch machines to accommodate this

Breaking DES

DES was designed for efficiency in early-70's hardware

Made it easy to build pipelined brute-force breakers in late-90's hardware



16 stages, tests 1 key per clock cycle

Breaking DES (ctd)

Can build a DES-breaker using

- Field-programmable gate array (FPGA), software-programmable hardware
- Application-specific IC (ASIC)

100 MHz ASIC = 100M keys per second per chip

Chips = $10 in 5K+ quantities

$50,000 = 500 billion keys/sec

= 20 hours/key (40-bit DES takes 1 second)

Breaking DES (ctd)

$1M = 1 hour per key ($^{1}/_{20}$ sec for 40 bits)

$10M = 6 minutes per key ($^{1}/_{200}$ sec for 40 bits)

(US black budget is ~$25-30 billion)

(distributed.net = ~70 billion keys/sec with 20,000 computers)

EFF (US non-profit organisation) broke DES in 2½ days

Amortised cost over 3 years = 8 cents per key

September 1998: German court rules DES “out of date and unsafe” for financial applications

Other Block Ciphers

AES

- Advanced Encryption Standard, replacement for DES
- 128 bit block size, 128/192/256 bit key, 10/12/14 rounds
- Non-Feistel structure
- Based on a sophisticated mathematical design
  - Easy to analyse security properties
  - Advances in mathematics may make it easier to analyse attacks

Blowfish

- Optimised for high-speed execution on 32-bit RISC processors
- 448 bit key, relatively slow key setup

Other Block Ciphers (ctd)

CAST-128

- Used in PGP 5.x, 128 bit key

GOST

- GOST 28147, Russian answer to DES
- 32 rounds, 256 bit key
- Incompletely specified

IDEA

- Developed as PES (proposed encryption standard), adapted to resist differential cryptanalysis as IPES, then IDEA
- Gained popularity via PGP, 128 bit key
- Patented

Other Block Ciphers (ctd)

RC2

- Companion to RC4, 1024 bit key
- RSADSI trade secret, reverse-engineered and posted to the net in 1996
- RC2 and RC4 had special status for US exportability
- Designed for 16-bit CPUs (8086), inefficient on more recent 32-bit RISC processors

Other Block Ciphers (ctd)

Skipjack

- Classified algorithm intended for the Clipper chip, declassified in 1998
- Very efficient to implement using minimal resources (e.g. smart cards)
- 32 rounds, breakable with 31 rounds
- 80 bit key, inadequate for long-term security

Triple DES (3DES)

- Encrypt + decrypt + encrypt with 2 (112 bits) or 3 (168 bits) DES keys
- After 1998, banking auditors were requiring the use of 3DES rather than DES based on precedents set in court cases

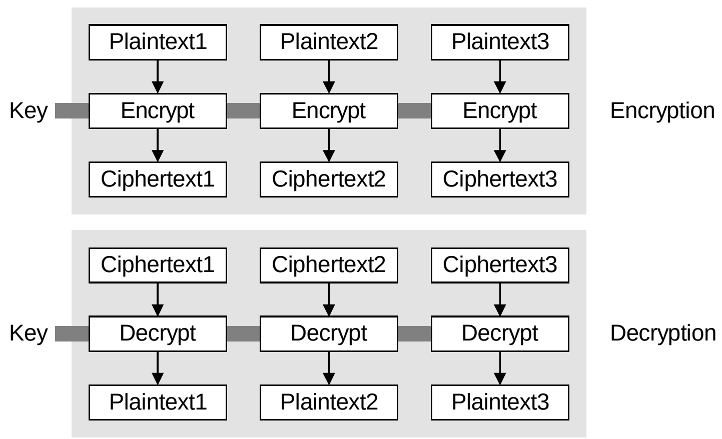
Other Block Ciphers (ctd)

Many, many others

- Fun to design, like wargames enthusiasts re-fighting historic battles
- No good reason not to use one of the above, proven algorithms

Using Block Ciphers: ECB

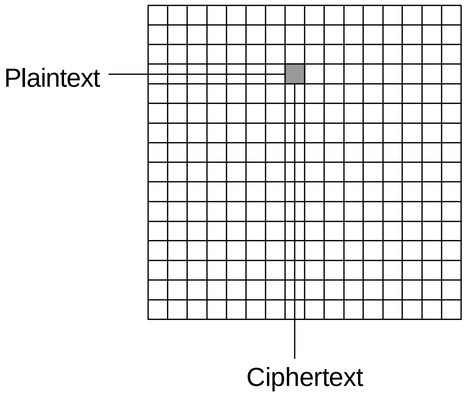
ECB, Electronic Codebook



Each block is encrypted independently

Using Block Ciphers: ECB (ctd)

Cipher acts as a 64-bit lookup table (electronic codebook)



Using Block Ciphers: ECB (ctd)

Original text

Deposit	$10,000	in acct.	number	12-3456-	789012-3

Intercepted encrypted form

H2nx/GHE	KgvldSbq	GQHbrUt5	tYf6K7ug	S4CrMTvH	7eMPZcE2

Second intercepted message

H2nx/GHE	5guZEHVr	GQHbrUt5	tYf6K7ug	Pts21LGb	a8oaNWpj

Cut and paste blocks with account information

H2nx/GHE	5guZEHVr	GQHbrUt5	tYf6K7ug	S4CrMTvH	7eMPZcE2

Decrypted message will contain the attacker’s account — without them knowing the encryption key

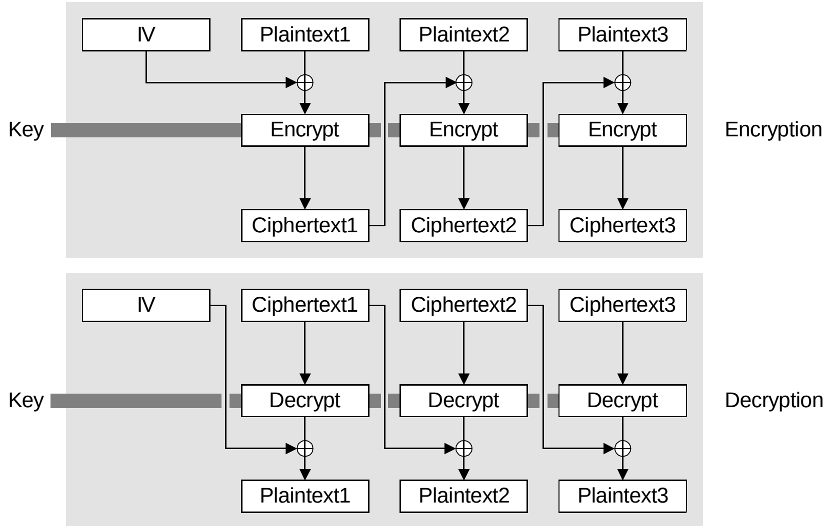
Using Block Ciphers: CBC

To protect against ECB-mode attacks, need to

- Chain one block to the next to avoid cut & paste attacks
- Randomise the initial block to disguise repeated messages
  – Inject initial randomness by prepending an Initialisation Vector (IV)

Using Block Ciphers: CBC (ctd)

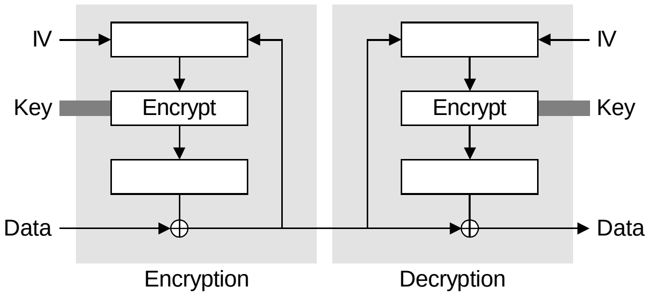
CBC, cipher block chaining, with IV for randomisation



Using Block Ciphers: CFB

Both ECB and CBC operate on entire blocks

CFB (ciphertext feedback) operates on bytes or even bits



This converts a block cipher to a stream cipher (with the accompanying vulnerabilities)

Using Block Ciphers: Other Modes

None of these modes provide integrity protection

- Chaining modes like CBC and CFB recover after one corrupted data block
  - This is a feature, since it provides error recovery

Various combined encryption + integrity-protection modes have been proposed

- All the ones that are as fast as just encryption are patented
- All the ones that aren’t patented aren’t much quicker than separate encryption + MAC
  - WPA uses one of these slow-but-unencumbered modes, AES-CCM

Relative Performance

Fast

RC4

AES, Blowfish, CAST-128

Skipjack

DES, IDEA, RC2

3DES, GOST

Slow

Typical speeds

- RC4 = Tens of MB/second
- 3DES = MB/second

Public Key Encryption

How can you use two different keys?

- One is the inverse of the other:

  key1 = 3, key2 = 1/3, message M = 4

  Encryption: Ciphertext $C = M \times key1$
  $= 4 \times 3$
  $= 12$

  Decryption: Plaintext $M = C \times key2$
  $= 12 \times 1/3$
  $= 4$

One key is published, one is kept private → public-key cryptography, PKC

Example: RSA

n, e = public key, n = product of two primes p and q

d = private key

Encryption: $C = M^e \bmod n$

Decryption: $M = C^d \bmod n$

p, q = 5, 7

$n = p \times q$
$= 35$

e = 5

$d = e^{-1} \bmod ((p-1)(q-1))$
$= 5$

Example: RSA (ctd)

Message M = 4

Encryption: $C = 4^5 \bmod 35$

$= 9$

Decryption: $M = 9^5 \bmod 35$

$= 59049 \bmod 35$

$= 4$

(Use mathematical tricks otherwise the numbers get dangerous)

Public-key Algorithms

RSA (Rivest-Shamir-Adelman), 1977

- Digital signatures and encryption in one algorithm
- Private key = sign and decrypt
- Public key = signature check and encrypt

DH (Diffie-Hellman), 1976

- Key exchange algorithm

Elgamal

- DH variant, one algorithm for encryption, one for signatures
- Attractive as a non-patented alternative to RSA (before the RSA patent expired)

Public-key Algorithms (ctd)

DSA (Digital Signature Algorithm)

- Elgamal signature variant, designed by the NSA as the US government digital signature standard
- Intended for signatures only, but can be adapted for encryption

DH, DSA, and Elgamal are all based on the discrete logarithm problem (DLP)

- Keys are interchangeable across DLP algorithms

All have roughly the same strength

- 512 bit key is marginal
- 1024 bit key is recommended minimum size
- 2048 bit key is better for long-term security

Using PKCs

PKCs are advanced mathematics, not just an X : Y mapping like a block cipher

- Can be attacked using mathematics
- Need to take special care in their use to avoid problems

Example: RSA

- Encrypt the same message to 3 people when e = 3
  - Recover message using the Chinese Remainder Theorem
- Sign a smooth (product of small primes) number
  - Allows forgery of signatures on other values
- Encrypt a guessable message
  - Allows message recovery through trial encryption with the public key

Using PKCs (ctd)

Countermeasures

- Pad the hash to be signed on the left with zeroes
  - Hash is small and likely to be smooth
- Pad the hash to be signed on the right with zeroes
  - Merely multiplies the hash by 2^n
- Pad the hash to be signed on the right with random data
  - Defeat with cube root attack (assuming e = 3)
- Many more similar pitfalls

PKCS

Public-key Cryptography Standard

- PKCS #1 covers safe use of RSA

RSA encryption

0	1	>= 8 nonzero random	0	Data

RSA signing

0	2	>= 8 bytes 0xFF	0	Data

- 0 guarantees value < modulus
- 1 or 2 distinguishes signed from encrypted data
  - RSA duality, signing = decryption
- Encryption padding produces a non-guessable message, ensures that each message is different, the message isn't a small value, etc
- Signature padding ensures the message isn't a small value, etc
- 0 delimits the end of the padding

DLP Algorithms

Need to be very careful with key generation

- Malicious user can generate booby-trapped keys
- DSA kosherizer and Lim-Lee algorithm guarantee verifiably safe keys

Incautious use of DLPs has the tendency to leak key bits

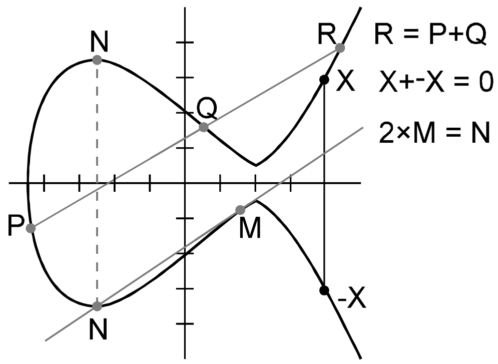
- RSA can do this too under some circumstances

Need to be *very* careful to apply PKCs correctly

- *Never* use raw RSA, DSA, DH, Elgamal, …
- Chances are you'll be using them incorrectly
  - "Evil will always triumph over good because good is dumb" — Dark Helmet

Elliptic Curve Algorithms

Use mathematical trickery to speed up public-key operations



Elliptic Curve Algorithms (ctd)

Now we can add, subtract, etc. So what?

- Calling it "addition" is arbitrary, we can just as easily call it multiplication
- We can now move (some) conventional PKCs over to EC PKCs (DSA → ECDSA)

Now we have a funny way to do PKCs. So what?

- Breaking PKCs over elliptic curve groups is much harder than beaking conventional PKCs
- We can use shorter keys that consume less storage space

Advantages/Disadvantages of ECC's

Advantages

- Sometimes useful in smart cards because of their low storage requirements

Disadvantages

- New, details are still being resolved
  - Many ECC techniques are still too new to trust
- Almost nothing uses or supports them
- No more efficient than standard algorithms like RSA
- ECCs are a minefield of patents, pending patents, and submarine patents

Recommendation: Don't use them unless you really need the small key size

Key Sizes and Algorithms

Conventional vs. public-key vs. ECC key sizes

Conventional	Public-key	ECC
(40 bits)	—	—
56 bits	(400 bits)	—
64 bits	512 bits	—
80 bits	768 bits	—
90 bits	1024 bits	160 bits
112 bits	1792 bits	195 bits
120 bits	2048 bits	210 bits
128 bits	2304 bits	256 bits

(Your mileage may vary)

Key Sizes and Algorithms (ctd)

However

- Conventional key is used once per message
- Public key is used for hundreds or thousands of messages

A public key compromise is much more serious than a conventional key compromise

- Compromised logon password, attacker can
  - Delete your files
- Compromised private key, attacker can
  - Drain credit card
  - Clean out bank account
  - Sign contracts/documents
  - Identity theft

Key Sizes and Algorithms (ctd)

512 bit public key vs. 40 bit conventional key is a good balance for weak security

Recommendations for public keys:

- Use 512-bit keys only for micropayments/smart cards
- Use 1K bit key for short-term use (1 year expiry)
- Use 1.5K bit key for longer-term use
- Use 2K bit key for certification authorities (keys become more valuable further up the hierarchy), long-term contract signing, long-term secrets

The same holds for equivalent-level conventional and ECC keys

Hash Algorithms

Reduce variable-length input to fixed-length (usually 128 or 160 bit) output

Requirements

- Can't deduce input from output
- Can't generate a given output (CRC fails this requirement)
- Can't find two inputs that produce the same output (CRC also fails this requirement)

Used to

- Produce a fixed-length fingerprint of arbitrary-length data
- Produce data checksums to enable detection of modifications
- Distil passwords down to fixed-length encryption keys

Also called message digests or fingerprints

MAC Algorithms

Hash algorithm + key to make the hash value dependant on the key

Most common form is HMAC (hashed MAC)

hash(key, hash(key, data))

- Key affects both the start and the end of the hashing process
- Having it at only one point would allow extension attacks

Naming: hash + key = HMAC-hash

MD5 → HMAC-MD5

SHA → HMAC-SHA

Recent attacks on MD5, SHA-1 don't affect HMAC form

Algorithms

MD2: 128-bit output, deprecated

MD4: 128-bit output, broken

MD5: 128-bit output, weaknesses

SHA-1: 160-bit output, NSA-designed US government secure hash algorithm, companion to DSA

SHA-2: Extension of SHA-1 design to larger output sizes

RIPEMD-160: 160-bit output

HMAC-MD5: MD5 turned into a MAC

HMAC-SHA: SHA-1 turned into a MAC

Pseudorandom Functions

Universal impedance-matcher for security algorithms

Used to transform one or more input values to a random (but input-dependent) output value

- Generate arbitrary-length pseudorandom sequences from fixed-length seeds
- Example: Convert a password and salt to a 3DES key

No standards for these

- Cryptographers: It's simple, just use HMAC, QED. Implementers: How should we use HMAC?
- No analysis of requirements
- Little security analysis

Pseudorandom Functions (ctd)

Everyone invents their own

- SSL/TLS

  out(0) = HMAC(key, HMAC(key, seed) || seed)

  out(n) = HMAC(key, out(n-1) || seed)

  – TLS uses dual HMAC-MD5 and HMAC-SHA1 XOR'd together in case one is found to be weak
- SSH

  out(0) = hash(key || exchange hash || session ID)

  out(1) = hash(key || exchange hash || out(0))

Pseudorandom Functions (ctd)

- IPsec
  – No consistency, whole range of ad hoc PRFs built using HMAC or raw hashes
    – See the IPsec section of the tutorial

  – One example: Encryption key calculation

  out(0) = HMAC(Ne_i, 0)

  out(n) = HMAC(Ne_i, out(n-1))
- PGP

  out(0) = hash(salt || password)

  out(n) = hash(n × '0' || salt || password)

Pseudorandom Functions (ctd)

- S/MIME (PKCS #5v2, PBKDF2)

  out(0) = HMAC(password, salt || ‘00000001’) XOR
  HMAC(password, previous-out) XOR
  …

  out(n) = HMAC(password, salt || ‘0000000n’) XOR
  HMAC(password, previous-out) XOR
  …

  – Sound approach, XOR protects against collapsing everything down to a single iteration