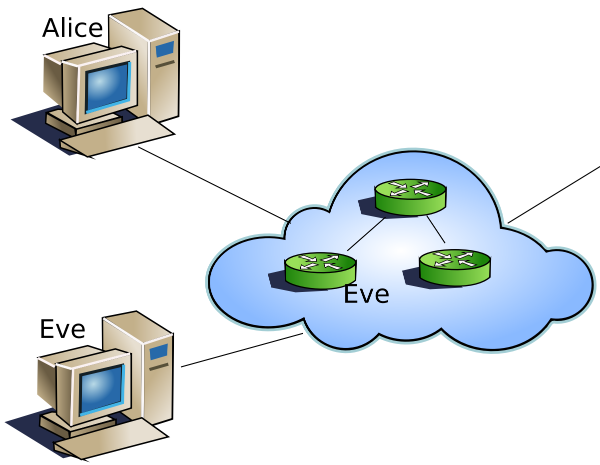
Introduction to Network Security and Cryptography

Simon Foley,
Department of Computer Science,
University College Cork

September 7, 2015

Network Security Challenges

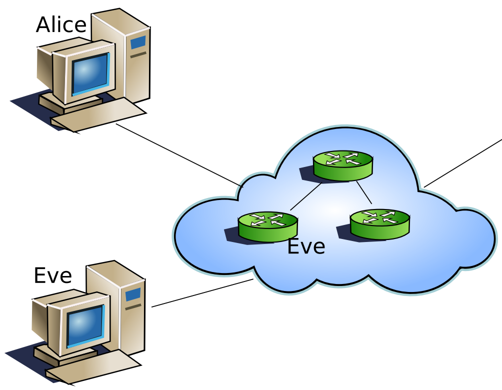
Some Basic Problems in the Network



An untrusted public network.

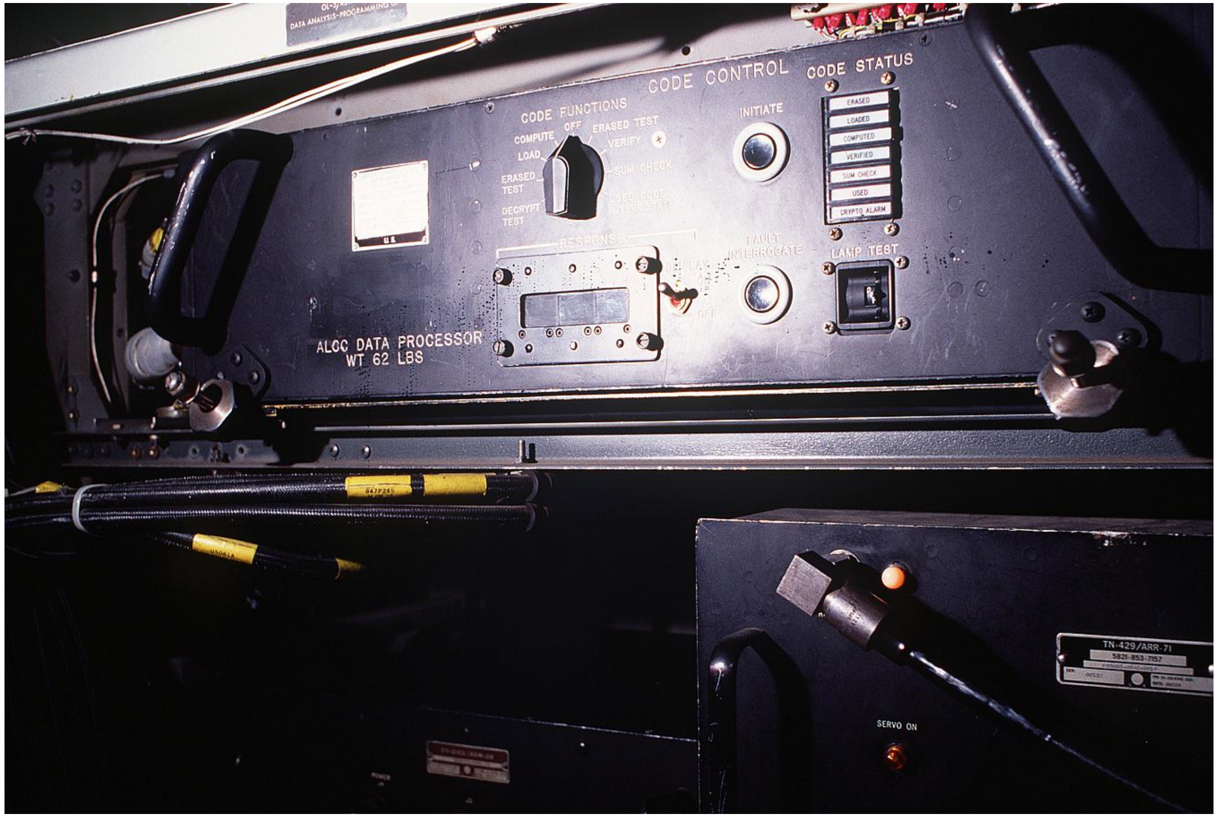
Problems: Eve eavesdrops on network packets, Eve modifies network packets, Eve injects messages pretending to come from Alice/Bob, Eve floods the network with packets, Alice repudiates a message, Eve gains access to Alice's system by exploiting some flaw. Alice repudiates a message she sent, Bob denies receipt of a message, Alice distrusts Bob, Alice distrusts the system she uses, Everyone distrusts the network, . . .

Some Basic Security Definitions



Principals represent active entities that wish to communicate: users, applications, servers, workstations, smart-sphones, smart-cards, routers, etc.

- ☐ *Confidentiality* - Data cannot be read by unintended recipients;
- ☐ *Integrity* - Data cannot be altered without detection;
- ☐ *Availability* - Data and resources are accessible and usable on demand by authorised entities.
- ☐ *Data Origin Authentication* - Data attributed to correct originator; (non-repudiation: who cannot disown it).



Close-up view of the airborne launch control system aboard an EC-135 Stratolifter "Looking Glass" aircraft of the 2nd Airborne Command and Control Squadron, 55th Strategic Reconnaissance Wing. The system decodes launch instructions from an encrypted tape and after two missile launch officers turn separate keys, it transmits a launch message to a Minuteman III missile housed in a silo [03/22/1991]

[http://research.archives.gov/description/6472193]

Very Basic Cryptography

A cryptographic cipher is a pair of Encrypt and Decrypt algorithms such that given *plaintext* P, encryption key K_1 and decryption key K_2 then

$$D(K_2, E(K_1, P)) = P$$

- ☐ In absence of knowledge about K_2, it must be not be feasible to recover P from the ciphertext $E(K_1, P)$.
- ☐ Given P and $E(K_1, P)$, it must not be feasible to recover K_1

Note that the *plaintext* P can be any data, including human readable text.

We call $E(K_1, P)$ the *ciphertext*.

Symmetric Cryptography: $K1 = K2$; (we'll study this first)

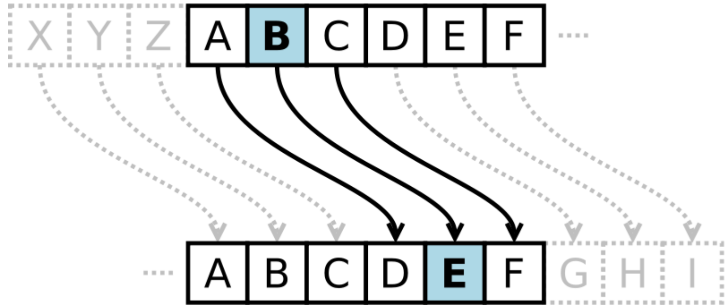
Asymmetric Cryptography: $K1 \neq K2$

Elementary Ciphers: Caesar Cipher

A *substitution cipher* used by Julius Caesar to communicate with his generals.

cipherChar=(plain char + 'd') mod 26

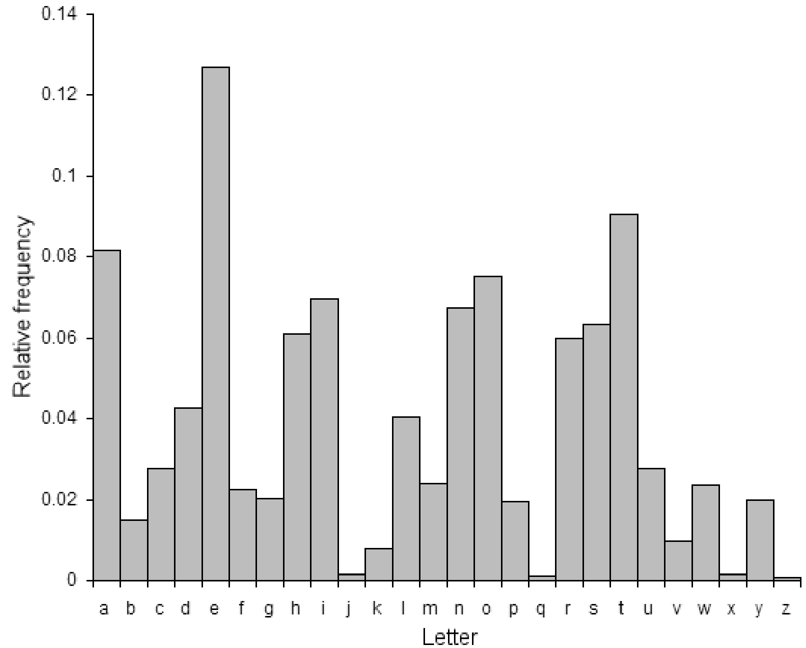
'`cat`' encrypted as '`fdw`'



Rekey: when Agustus Caesar took the throne, he changed the *key* 'd' to 'e'

`ROT13` cipher character = (plain character + '13) mod 26

Easy to break caesar cipher by studying letter frequencies (a form of cryptanalysis): the caesar cipher simply shifts the distribution.



Ciphertext = `dwwdfndwgdzq`
Plaintext =

Vigenère Cipher

The Vigenère substitution cipher attempts to thwart letter-frequency analysis by changing the substitution key (eg 'd') for each character. The substitution key is selected from a longer secret key.

cipher character = (plain character + key character) mod 26

Suppose the secret key was the string 'mykey', then encrypting:

Plaintext = `attackatdawn`
+
key = `mykeymykeymy`
=
Ciphertext = `mrdeawydhyil`

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

While better than a Caesar cipher, the repeating key means that repeating patterns will eventually show in a long fragment of text.

One Time Pad

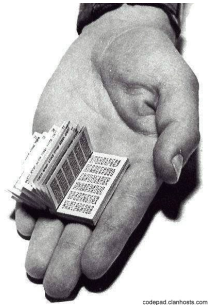
Use Vigenère with a key that is as long as the plaintext and never repeats.

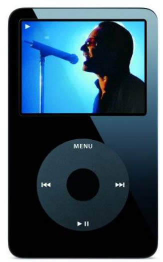
A truly key random implies we have what's called *perfect secrecy* (nothing about the plaintext/key can be determined from ciphertext)

In practice, the key and plaintext are sequences of binary values and encryption/decryption is implemented as bitwise XOR $\oplus$ operation.

$$
\begin{array}{ll|ll}
\text{Plaintext} & = 1100001111100 & \text{Ciphertext} & = 1101110110100 \\
\oplus & & \oplus & \\
\text{key} & = 0001111001000 & \text{key} & = 0001111001000 \\
= & & = & \\
\text{Ciphertext} & = 1101110110100 & \text{Plaintext} & = 1100001111100
\end{array}
$$

Example Keys (one time pads):



One Time Pad Problems

- Key ('pad') size, storage and distribution;
- Never re-use a key (we'll see why later).
- Difficult to synchronize the key/text between sender and receiver.
- 'attack in depth' on integrity of message: the recipient has no reliable way of knowing whether a message has been interfered with.

plain	`mydear` ...	*guess plaintext*	`sodoff` ...
key	`abxkie` ...	*determine key*	`abxkie` ...
cipher	`naaoiv` ...	*from ciphertext*	`spaynj` ...

See `http://www.nsa.gov/about/cryptologic_heritage/museum/` for more history.

Modern Cryptography

Symmetric and Asymmetric Ciphers

- **Data Encryption Standard (DES) 1970's.**
  A symmetric block cipher that uses a 56 bit key.
  A weak scheme given current advances on brute-force techniques.
  Triple-DES variant (used widely by banks) is currently considered safe.
- **Advanced Encryption Standard (AES) 2000's**
  A symmetric block cipher that can use 128, 192 or 256 bit keys.
  The 'new' standard for commercial grade symmetric ciphers.
- **RSA Public Key Cipher 1970's**
  An asymmetric cipher based on some special properties of numbers.
  Uses two keys: one for encryption and the other for decryption.
  Recommended to be at least 1024 bits long. Computationally hard to discover one key from the other

Details on these, and other cryptographic schemes, later.

Recall the requirement for a good cipher:

- Given P and $E(K, P)$, it must not be *feasible* to recover K

Suppose a malicious individual has a copy of some plaintext P and corresponding cipher text C (encrypted under secret key K).

If the number of possible key values is known to be small then it may be possible to test every possible key K until we have $E(K, P) = C$.

This is called a *known-plaintext* brute force attack.

In simple terms, A key of size n bits has 2^n possible key values and the keys must be large (number of bits) enough to make 'brute-force' attack impractical. The computational effort required to find a key in this way is exponential in the size of the key.

Consider a processor that can test 1 million keys per second. Time to find correct key: 56bit key (DES) 2,000 years; 128bit key (eg, AES) 10^{25} years.

Recommended key-size depends on crypto algorithm used, but typically anything bigger than 128 bits is OK for a symmetric cipher.

Given key K, we have plaintext P and ciphertext $C = E(K, P)$

- If K is a one-bit key, then a brute force search requires at most 2 tests: $C \stackrel{?}{=} E(0, P)$ and $C \stackrel{?}{=} E(1, P)$
- If K is a two-bit key, then a brute force search requires at most 4 tests: $C \stackrel{?}{=} E(00, P)$, $C \stackrel{?}{=} E(01, P)$, $C \stackrel{?}{=} E(10, P)$ and $C \stackrel{?}{=} E(11, P)$
- If K is a three-bit key, then a brute force search requires at most 8 tests: $C \stackrel{?}{=} E(000, P)$, ..., $C \stackrel{?}{=} E(111, P)$
- ...
- if K is an n-bit key then brute force search requires at most 2^n tests;

Adding an extra bit to the length of a key doubles the size of the keyspace/work required to brute force the key: $2^{n+1} = 2 \times 2^n$.

Brute Force Attacks: EFF Deep Crack

Electronic Frontier Foundation (EFF) built (1998) a parallel machine to brute-force search the entire 2^{56} key space of DES.

Built to make the case that a 56-bit key is too small.

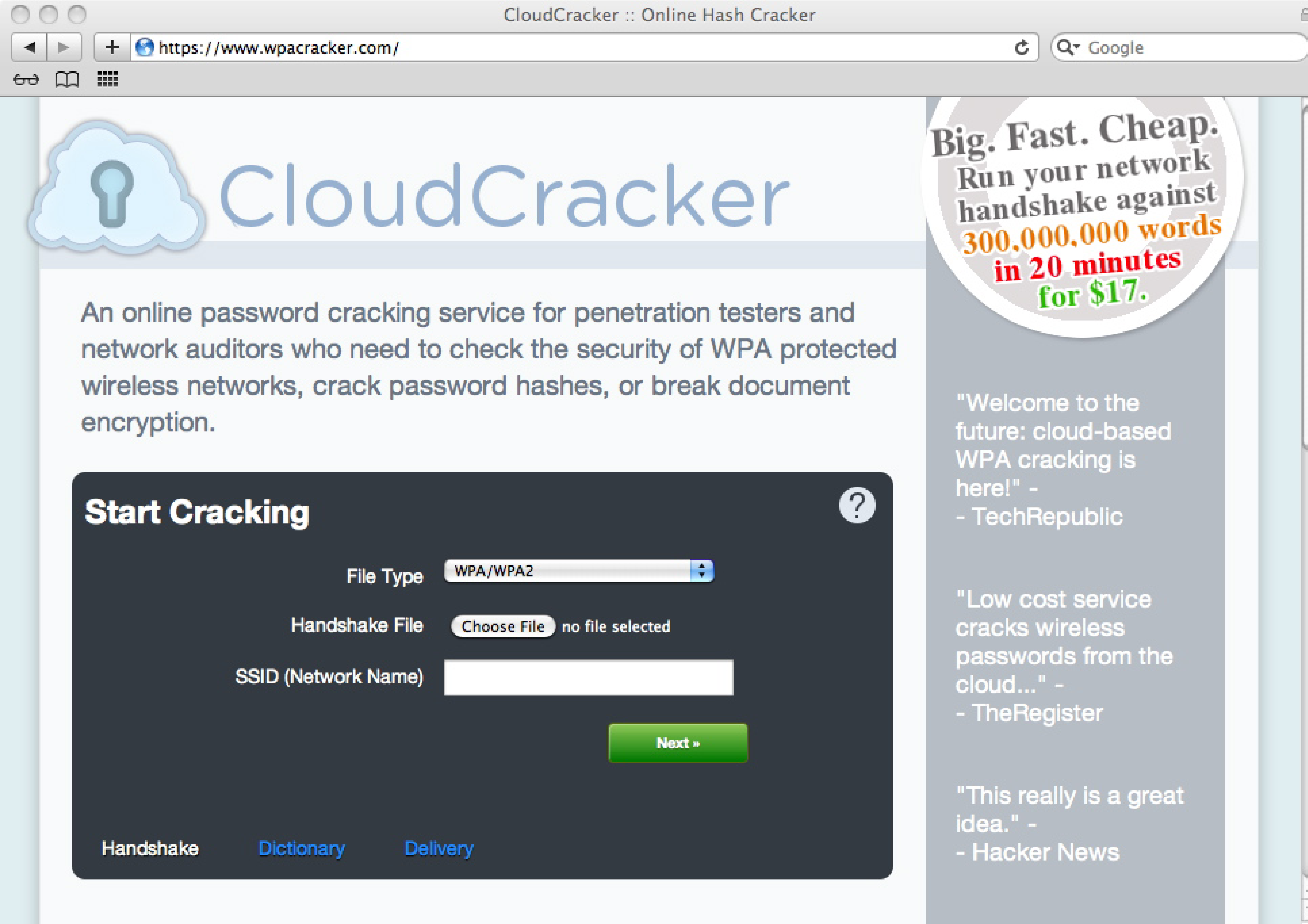
Machine cost US$250,000 (1998); performs 40 billion key tests per second and finds a key in under 5 days!

Key Size	Possible combinations
1-bit	2
2-bit	4
4-bit	16
8-bit	256
16-bit	65536
32-bit	4.2×10^{9}
56-bit (DES)	7.2×10^{16}
64-bit	1.8×10^{19}
128-bit (AES)	3.4×10^{38}
192-bit (AES)	6.2×10^{57}
256-bit (AES)	1.1×10^{77}

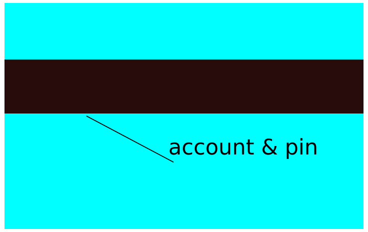
Key size	Time to Crack
56-bit	399 seconds
128-bit	1.02×10^{18} years
192-bit	1.872×10^{37} years
256-bit	3.31×10^{56} years

Timing based on world's fourth fastest computer (2013) K-Computer: 10.51 Petaflops $= 10.51 \times 1015$ Flops

See also: `http://www.copacobana.org` FPGA-based cracking hardware. `http://www.wpacracker.com` cloud-based cracking services.


CloudCracker :: Online Hash Cracker
https://www.wpacracker.com/
Google
CloudCracker
Big. Fast. Cheap.
Run your network handshake against
300,000,000 words
in 20 minutes
for $17.
An online password cracking service for penetration testers and network auditors who need to check the security of WPA protected wireless networks, crack password hashes, or break document encryption.
Start Cracking
?
File Type
WPA/WPA2
Handshake File
Choose File
no file selected
SSID (Network Name)
Next »
Handshake
Dictionary
Delivery
"Welcome to the future: cloud-based WPA cracking is here!" -
- TechRepublic
"Low cost service cracks wireless passwords from the cloud..." -
- TheRegister
"This really is a great idea." -
- Hacker News

Applying Crypto in practice: Bank ATM



Bank issues ATM card and PIN to customers;
ATM card used to identify customer;
Card and PIN used to *authenticate* customer;
Bank staff should not have access to PIN

The pin is a secret shared between the bank (ATM) and the customer. It should not be known by anybody else (including bank staff). The bank would like to be sure that the pin is properly protected on the card.

Which implementation Strategy works?

- ☐ Store [acctid,pin] on card as cleartext; acctid provides customer id and ATM checks pin provided by user against pin stored on card.
- ☐ Store $[acctid, E(K_B, pin)]$ on card; K_B secret key known only to Bank.
- ☐ Store $[E(K_B, (acctid : pin))]$ on card.

Applying Crypto in practice: Bank ATM

In practice, store *acctid,offset* is stored on the back of the card.

ATM adds first 4 digits of $E(K_B, \textit{acctid})$ to *offset* and compares against pin.

Security depends on keeping K_B secret: K_B should not be accessible by bank staff.

In practice K_B accessible only via 'tamper-resistant' cryptographic hardware modules which perform the cryptographic operations.

Bank ATM Skimming

More examples: `http://krebsonsecurity.com/tag/atm-skimmer`.

Clues to reducing the keyspace

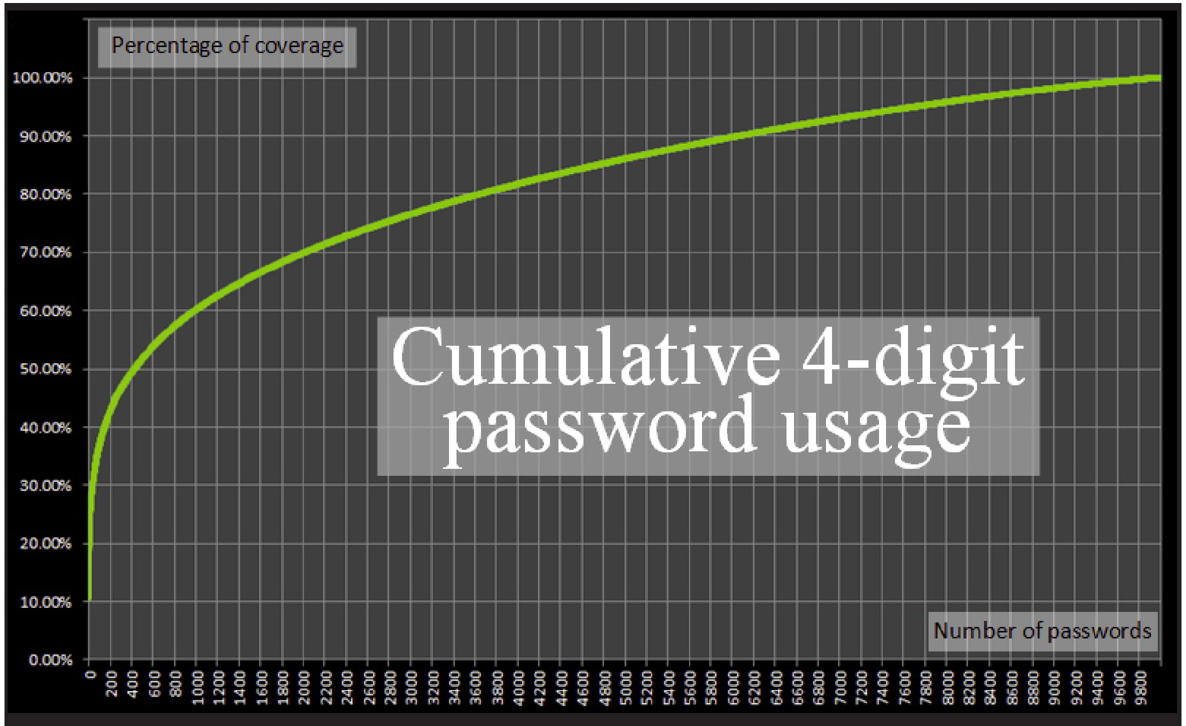
Is there a way to reduce the number of keys/pins to be tested/brute forced?

This may not be a problem for an ATM machine since different users have different PINs (but are some numbers more common than others?)

On some security systems the same PIN is used over and over again. For example, an alarm panel:

The PINs people choose

	PIN	Freq
#1	1234	10.713%
#2	1111	6.016%
#3	0000	1.881%
#4	1212	1.197%
#5	7777	0.745%
#6	1004	0.616%
#7	2000	0.613%
#8	4444	0.526%
#9	2222	0.516%
#10	6969	0.512%
#11	9999	0.451%
#12	3333	0.419%
#13	5555	0.395%
#14	6666	0.391%
#15	1122	0.366%
#16	1313	0.304%
#17	8888	0.303%
#18	4321	0.293%
#19	2001	0.290%
#20	1010	0.285%



Statistically, *one third* of all codes can be guessed by trying just 61 distinct combinations!

http://www.datagenetics.com/blog/september32012/

Last Updated: Monday, 25 June 2007, 05:35 GMT 06:35 UK

The man who invented the cash machine

By Brian Milligan
Business reporter, BBC News

"They're clever scoundrels," fumes John Shepherd-Barron at his remote farmhouse in northern Scotland. He is referring to the seals which are raiding his salmon farm and stealing fish.

"I invented a device to scare them off by playing the sound of killer whales, but it's ended up only attracting them more."

But failure with this device is in contrast to the success of his first and greatest invention: the cash machine.

The world's first ATM was installed in a branch of

One by-product of inventing the first cash machine was the concept of the Pin number.

Mr Shepherd-Barron came up with the idea when he realised that he could remember his six-figure army number. But he decided to check that with his wife, Caroline.

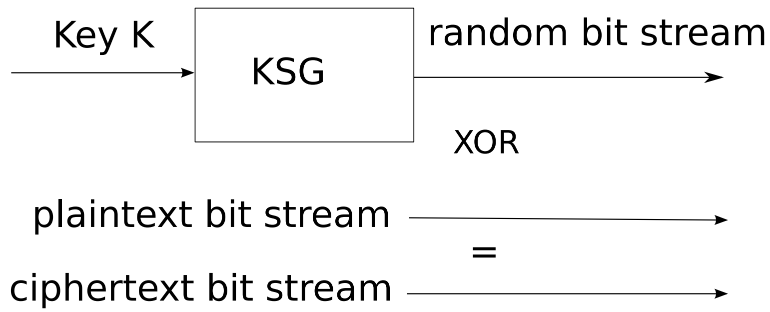
"Over the kitchen table, she said she could only remember four figures, so because of her, four figures became the world standard," he laughs.

VIDEO AND AUDIO NEWS
Cash machine's inventor recalls early days
Watch

SEE ALSO
- More cash machines for the poor
  19 Jun 07 | Business
- Tricky questions at cashpoints abroad
  22 May 07 | Business
- Q&A: Cash machine charges
  28 Jul 06 | Business
- Charging ATMs 'must be labelled'
  31 Mar 05 | Business

RELATED INTERNET LINKS
- Barclays

Stream Ciphers and Key-Stream Generators



Key-stream generator (KSG) generates a long sequence of random-looking bits, given some initial random seed (key).

Could I use a Pseudo Random Number Generator to implement KSG?

For example, given (public) constants c_1, c_2 and N then the i^{th} random number X_i is computed as

$$X_i = (c_1 \times X_{i-1} + c_2) \bmod N$$

where X_{i-1} is previous random value generated; the PRNG is seeded by a random value X_0.

Even though I know the KSG algorithm and have considerable quantity of generated key stream, I should not be able to predict any more of it, assuming I don't know the key.

A standard PRNG does not have this property.

Suppose that each X_i in the PRNG is 32 bits wide and the attacker has the first 16 bits of the plaintext stream P_0 and its corresponding ciphertext C_0. Since $C_0 = P_0 \oplus X_0$ (bitwise xor with first 32 bits of keystream), then given P_0 and C_0 the attacker can easily compute $X_0 = P_0 \oplus C_0$. We want to be sure the attacker cannot use X_0 to determine any further keystream X_i.

Stream cipher applications: implement bitwise XOR in hardware. Satellite TV (eg BSkyB) signals. PKZIP (very poor cipher).

GSM telephones used a propriety stream cipher A5/1 (64 bit key) to encrypt link between telephone and base station. It was reverse engineered and an attack carried out (requiring 40-200 bytes of known plaintext, time complexity 2^{27}).

The Misuse of RC4 in Microsoft Word and Excel [Wu, 2005]

An old version of Microsoft Office allowed documents to be encrypted using the RC4 stream cipher, initialized using a key/password provided by the user.

RC4 is a secure KSG. However, it can be used incorrectly....

Let $\oplus$ represent bitwise XOR of data and let RC4(K) represent the key steam generated by RC4 initialized by secret password K.

Suppose Alice saves her document P_1 (plaintext), protected using key K. Ciphertext $C_1 = P_1 \oplus RC4(K)$ is saved to disk.

Suppose that Alice edits her document, creating P_2, which is saved as $C_2 = P_2 \oplus RC4(K)$, using the same key.

Its easy to spot differences between the encrypted documents C_1 and C_2.

Its easy to spot differences in the encrypted documents

P_1

```
000a00  41 6E 74 69 2D 76 69 72  75 73 20 72 65 73 65 61  Anti-virus resea
000a10  72 63 68 65 72 73 20 66  72 6F 6D 20 53 79 6D 61  rchers from Syma
000a20  6E 74 65 63 20 79 65 73  74 65 72 64 61 79 20 73  ntec yesterday s
000a30  70 6F 74 74 65 64 20 74  68 65 20 66 69 72 73 74  potted the first
000a40  20 76 69 72 75 73 20 63  61 70 61 62 6C 65 20 6F   virus capable o
000a50  66 20 69 6E 66 65 63 74  69 6E 67 20 36 34 2D 62  f infecting 64-b
000a60  69 74 20 57 69 6E 64 6F  77 73 20 73 79 73 74 65  it Windows syste
000a70  6D 73 2E 0D 00 00 00 00  00 00 00 00 00 00 00 00  ms..............
```

Fig. 1. Binary format of the original document (unencrypted)

C_1

```
000a00  3E 57 FB B6 64 22 4A CA  3A 74 40 E7 1D 57 C6 DB  >W..d"J.:t@..W..
000a10  A3 88 21 53 F2 DB 3B 64  21 2A AD DD A8 7C 35 85  ..!S..;d!*...|5.
000a20  9B ED E5 F6 68 9A 35 47  68 89 9A ED 44 AE BF 08  ....h.5Gh...D...
000a30  D2 D5 CB 2B 0B 6B 45 4F  42 06 DC C6 C1 A5 81 B5  ...+.kEOB.......
000a40  AF 39 6F F1 1C 84 1F 88  B0 FD E1 09 D8 B9 E0 24  .9o............$
000a50  6C 1C 42 7C B7 D6 63 10  80 0B D5 B7 7F 01 6C 9B  l.B|..c.......l.
000a60  B8 4A F9 67 0D 27 FD 49  8E 98 76 9D C5 0F B0 E4  .J.g.'.I..v.....
000a70  AF 95 AC A2 5E 61 DD 9D  71 92 3A B9 40 AE CB F3  ....^a..q.:.@...
```

Fig. 2. Binary format of the original document (encrypted)

C_2

```
000a00  3E 57 FB B6 64 22 4A CA  3A 74 40 E7 1D 57 C6 DB  >W..d"J.:t@..W..
000a10  A3 88 21 53 F2 DB 3B 63  27 65 93 84 96 64 36 90  ..!S..;c'e...d6.
000a20  90 FA A0 EC 2D 90 24 51  6E 88 89 F0 05 A4 EF 14  ....-.$Qn.......
000a30  D6 CE DA 3B 4E 7B 0D 5E  0A 05 95 D2 DB A3 D2 B7  ...;N{.^........
000a40  E6 3D 73 F0 49 94 5E 9B  B0 EF EC 0E 94 B3 A6 6B  .=s.I.^........k
000a50  63 52 4D 77 B2 C7 69 0A  8E 45 84 A3 64 57 28 8D  cRMw..i..E.dW(.
000a60  F1 69 B0 5E 00 26 EE 55  D9 98 2F 9D C8 19 A9 F2  .i.^.&.U../.....
000a70  EC EB 82 AF 5E 61 DD 9D  71 92 3A B9 40 AE CB F3  ....^a..q.:.@...
```

Fig. 3. Binary format of the modified document (encrypted)

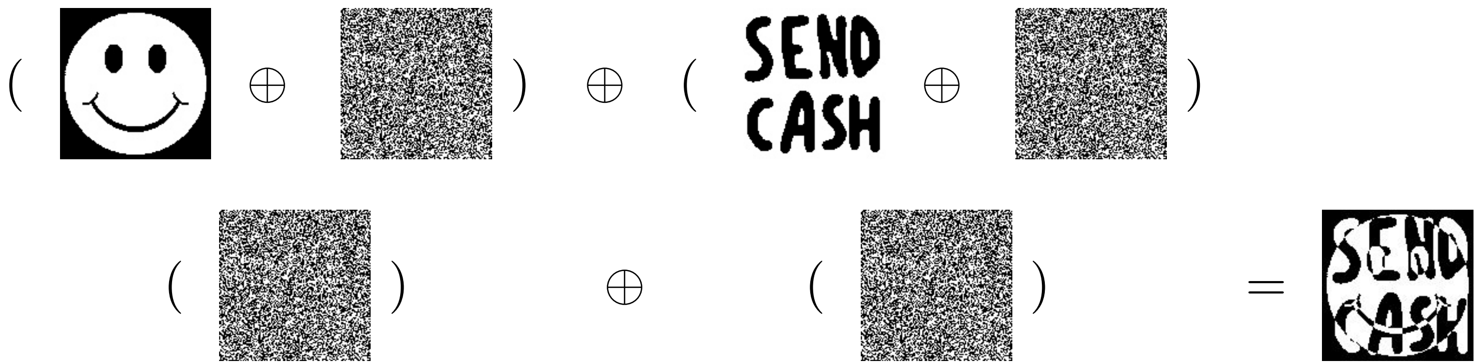
Stream Cipher Attack: problems with reusing key-streams

Suppose that a thief steals Alice's laptop (containing the ciphertext documents C_1 and C_2 above).

The thief extracts C_1 and C_2 and computes

$$\begin{aligned} C_1 \oplus C_2 &= (P_1 \oplus RC4(K)) \oplus (P_2 \oplus RC4(K)) \\ &= P_1 \oplus P_2 \end{aligned}$$

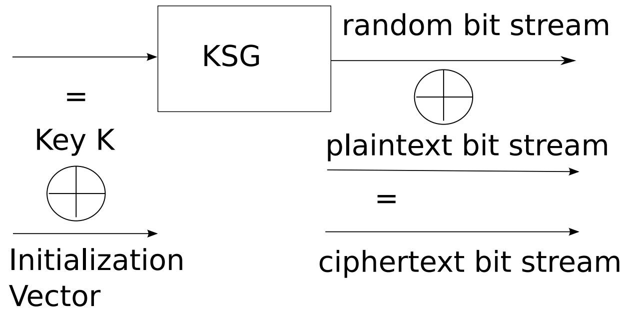
If the attacker knows P_1 then he can compute P_2. $P_1 \oplus P_2$ also tells the attacker something about the underlying plaintext. For example,



(see `http://www.cryptosmith.com/archives/70`)

The same keystream should not be used to encrypt more than one message.

This can be done by adding a different initialization vector to the key, each time it is used.



For example, given plaintext P and secret key K then a random initialization vector IV is generated and $C = RC4(K \oplus IV, P)$.

Note that the IV must also be sent/stored with C so that the recipient, knowing K, can recreate the key-stream in order to decrypt C to give P.

[recall that with a one-time pad we should never re-use the key]

Stream Cipher Initialization Vector

Suppose a 1-bit initialization vector is used then there are just two possible key streams and given any two ciphertexts there's a 50:50 chance that they use the same key stream. For example, encrypting different plaintext:

$C_1 = RC4(K \oplus 0, P_1)$, $C_2 = RC4(K \oplus 1, P_2)$, $C_3 = RC4(K \oplus 0, P_3)$, ...

In this case, the attacker waits for the IV collision (same value) and computes $C_1 \oplus C_3$. Need to be sure that initialization vector is large enough to ensure that the probability of reuse is low.

Wired Equivalent Privacy (WEP) is a protocol used in IEEE 802.11 that uses a stream cipher to secure connection between wireless devices.

In WEP the IV is only 24 bits, providing a basis for attack: the attacker collects encrypted frames (ciphertext plus random IV) until such time that he finds two frames C_1 and C_2 that have the same initialization vector IV. (note that a small IV is not the only vulnerability, and regardless, it is advised to use WPA2).

Like the one-time-pad, a stream cipher also does not provide integrity and the ciphertext is vulnerable to an attack in depth.

Transposition Ciphers

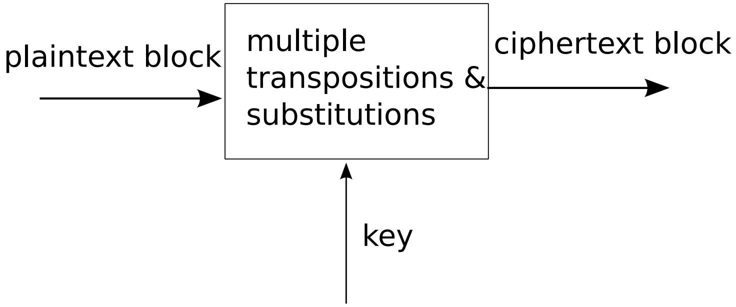
The letters stay the same but their positioning in the plaintext stream changes (anagrams).

$$\texttt{attackatdawn} \longrightarrow \left| \begin{array}{l} \texttt{atta} \\ \texttt{ckat} \\ \texttt{dawn} \end{array} \right| \longrightarrow \texttt{acdtkatawatn}$$

Plaintext is transposed one block at a time.

The *Block Ciphers* that we will use use combinations of transpositions and substitutions.

Symmetric Ciphers in Practice



Block cipher encrypts/decrypts data one block at a time.

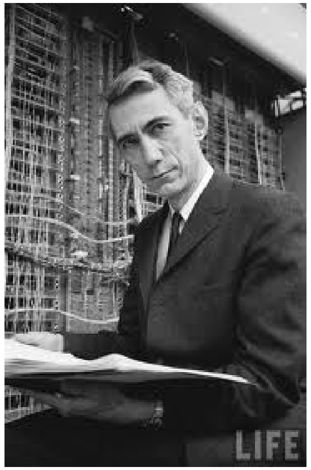
DES Block cipher (1977), 64 bit block size, 56 bit key.

AES Block cipher (2000), 128 bit block size, 128, 192 or 256 bit key.

Standards are a good thing.

Kerckhoffs' principle:
a cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

Shannon's maxim:
the enemy knows the system.



Multiple Encryption: encrypting the same block with different keys??

Double DES uses 2 keys: $C = E(k_1, E(k2, P))$. But is no more secure than single DES (meet in middle attack).

Triple DES: Encrypt-decrypt-encrypt: for DES, then the following is considered to provide equivalent of an 80-bit key.

$$
\begin{aligned}
C &= E(k_1, D(k_2, E(k1, P))) \\
P &= D(k_1, E(k_2, D(k1, C)))
\end{aligned}
$$

Scheme preserves compatibility with single-encryption: if $k1 = k2$ then $E(k_1, P) = E(k_1, D(k_2, E(k_1, P)))$, $D(k_1, C) = D(k_1, E(k_2, D(k_1, C)))$. Scheme used to improve DES for X9.17 and ISO8732 standards.

DESX uses 3 keys and requires only one round of DES:

$$C = k_1 \oplus E(k_2, P \oplus k_3)$$

Effective key length believed to be around 112 bits.

Moral: Never invent your own cipher scheme!

Meet in the Middle Attack on Double DES

Note symmetry in double DES: given $C = E(k_1, E(k2, P))$,

$$E(k_2, P) \quad = X = \quad D(k_1, C)$$

Known plaintext attack given P and C:

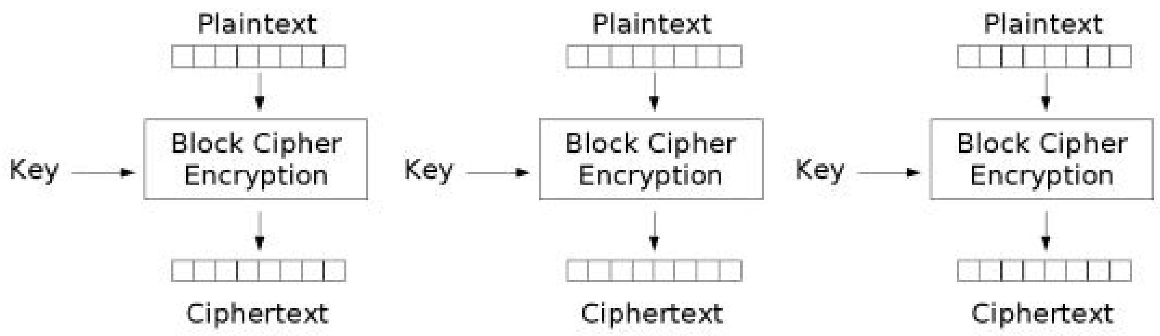
1. Encrypt P for all 2^{56} values of k_2 and store in a table indexed by $X = E(k_2, P)$.
2. Decrypt C for all 2^{56} values of k_1; as decryption is calculated, check result against table for a match; if match occurs, then test resulting pairs against the P,C pair.

While this is a *theoretical attack*—cost is 2×2^{56} crypto operations plus 2^{56} storage—it demonstrates that the strength of the cipher is not the sum of they key sizes as was originally thought.

Applicable to any double-encryption cipher.

Block Cipher Modes: Electronic Code Book

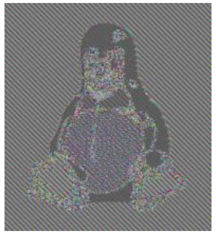
Encrypt each block independent of the other blocks.



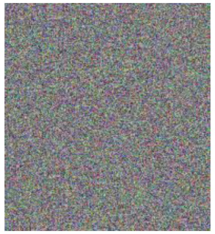
Electronic Codebook (ECB) mode encryption

Problem: plaintext patterns show through in ciphertext.

plaintext



ECB encrypted



CBC encrypted

Is this a Good Design?

Lecturer Simon shares a secret DES key K_{SE} with the UCC exams office server and submits exam results R using the following protocol.

$$Simon \rightarrow Exams : E_{DES\_ECB}(K_{SE}, R)$$

Results are formatted using a fixed length record:

StudentID	Name	ExamRslt	YW
Int (4bytes)	String (28bytes)	Int (4bytes)	Int (4bytes)
543	Allen	160	20
553	Brown	120	10
...	...	...	...
515	Smith	40	5

Each record fits into exactly four (DES) blocks, and the i^{th} student record starting at block $5i$ has the student grade in block $5i + 4$. The file is sorted by student name.

Is this a Good Design?

Lecturer Simon shares a secret DES key K_{SE} with the UCC exams office server and submits exam results R using the following protocol.

$$Simon \rightarrow Exams : E_{DES\_ECB}(K_{SE}, R)$$

Results are formatted using a fixed length record:

StudentID	Name	ExamRslt	YW
Int (4bytes)	String (28bytes)	Int (4bytes)	Int (4bytes)
543	Allen	160	20
553	Brown	120	10
...	...	...	...
515	Smith	40	5

Each record fits into exactly four (DES) blocks, and the i^{th} student record starting at block $5i$ has the student grade in block $5i + 4$. The file is sorted by student name.

Suppose that student Smith controls a router between Simon and the Exams Server. While he cannot decrypt the result-file, he knows that Allen is the best in the class and cut-pastes Smith's (encrypted) grade by Allen's (encrypted) grade in the file enroute.

Is this a Good Design?

A commercial email system once used DES ECB with a key given by the user as an 8 character password. The design was such that the last block of every (cleartext) email message was always a block of nulls (this acts as a recognizer for the end of message).

- Known Plaintext Brute Force Attack. Assume that users pick only lowercase letters for their passwords, then the number of possible keys (key-space) is 26^8. This means the effective key size $log_2(26^8)$ is only around 26 bits and easy breakable by brute force!
- Precomputation Dictionary Attack. An attacker pre-computes and builds a dictionary file of records $(k, E(k, nulls))$ based on words k from a dictionary. At some later date when given an encrypted email message, the attacker extracts the last block (encrypted nulls) and looks for a corresponding (poorly chosen) password from the dictionary.
- Integrity Attack: we can cut & paste encrypted blocks!

In general, ECB mode encryption should not be used.

Brute Force Dictionary Attack

The attacker knows plaintext $nulls$ and ciphertext $E(K, nulls)$.

Suppose the key-space for K is large then a conventional known-plaintext brute force attack is not feasible.

However, a naive user may have selected a word from a dictionary as their password/key.

Strategy: each time the attacker sees an encrypted block of nulls he then does a brute-force attack but only testing those keys that correspond to words in the dictionary.

While this is much more feasible than having to test the entire key-space, the attacker must repeat the attack (testing words from dictionary) for every email message that he sees and there is no guarantee that he will find a key for a given encrypted block.

Pre-computation Dictionary Attack

The attacker knows *every* encrypted email message has ciphertext block $E(K, nulls)$ (for plaintext $nulls$).

Build (pre-compute) a dictionary table $(w, E(w, nulls))$, for each word w in dictionary. This is done just once.

w	$E(w, nulls)$
Aachen	35423242
aardvark	73737422
...	...
zymurgy	635124534

Each time the attacker sees a block of nulls encrypted with a password he does not know, at the end of an encrypted email message, he checks to see whether it is in the dictionary table. If so, then the corresponding w is the password/key used.

The main cost of this attack is the pre-computation and storage of the table. Once that is done, the cost of attack per encrypted message is cheap: a single table lookup.

Pizzas to the Pentagon: different encrypted messages should look different

Also see `http://home.xnet.com/~warinner/pizzacites.html`

Adding randomness to the encrypted message

Recall our use of an Initialization vector with a stream cipher as a way to make messages encrypted with the same key look different.

For example, when encrypting a message (a series of blocks $P_1, P_2, \ldots$), Alice A might use the key $K \oplus IV$, where K is the shared secret and IV is a random initialization vector created for the message.

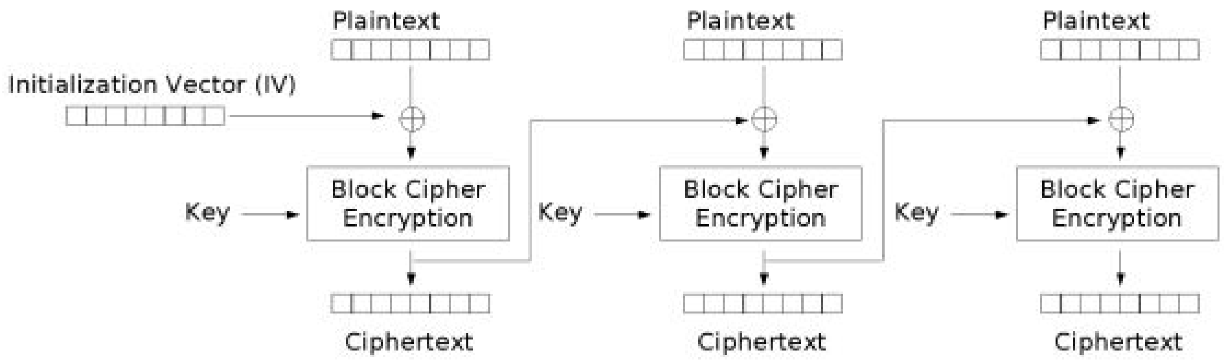
The IV is sent along with the cipher text to the recipient Bob B (who also knows secret K).

$$A \rightarrow B : IV, E_{ECB}(K \oplus IV, P_1), E_{ECB}(K \oplus IV, P_2), \ldots$$

Eve, listening in on the communication, will not be able to spot patterns between *different* messages, however, she will still be able to spot patterns within the *same* message (since the IV is the same for each block: recall the encryption of the Tux image)

Cipher Block Chaining mode generalizes this idea by XOR'ing each block of plaintext in a message with a different random value, ensuring that each cipher block within a message looks different.

Block Cipher Modes: Cipher Block Chaining



Cipher Block Chaining (CBC) mode encryption

$$
\begin{aligned}
C_0 &= E(K, P_0 \oplus IV) \\
C_1 &= E(K, P_1 \oplus C_0) \\
&\ldots \\
C_i &= E(K, P_i \oplus C_{i-1})
\end{aligned}
$$

$$
\begin{aligned}
P_0 &= D(K, C_0) \oplus IV \\
P_1 &= D(K, C_1) \oplus C_0 \\
&\ldots \\
P_i &= D(K, C_i) \oplus C_{i-1}
\end{aligned}
$$

Initialization vector sent in the clear along with ciphertext. Disguises patterns in plaintext; different IV ensures same messages look different.

Practicalities: Block Ciphers and Padding

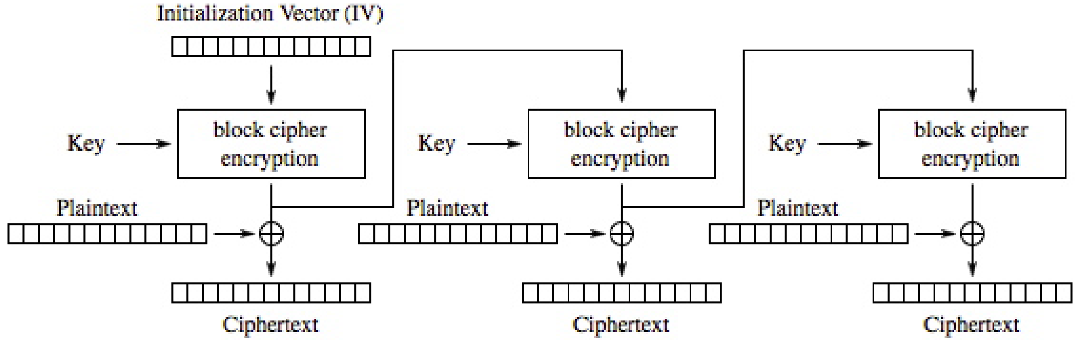
If the plaintext to be encrypted is not a multiple of the block size then the last block needs to be 'padded'. Padding using nulls or some other value will not work (why?).
PKCS5 (an RSA 'declared'/defacto standard) specifies that the unused space in a block should be filled with bytes containing the number of remaining bytes. For example, the following 8Byte blocks are padded:

•	•	•	•	•	3	3	3
•	•	•	•	•	•	•	1

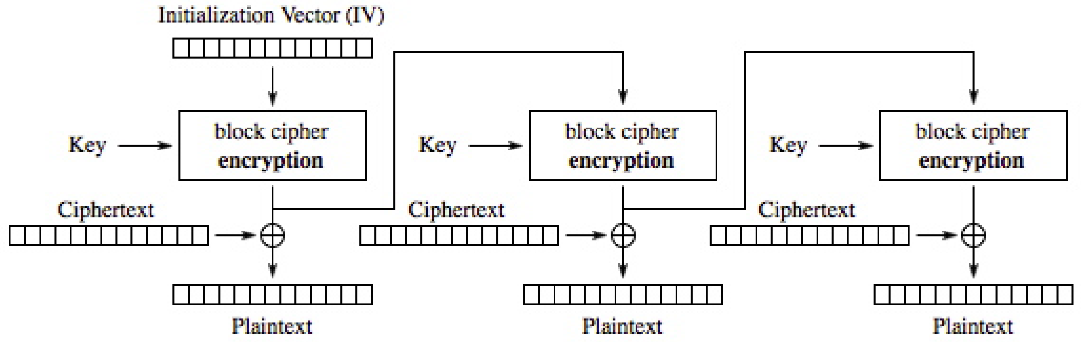
If the size of the plaintext is a multiple of the block size then we should append an additional final block, completely padded (otherwise the recipient has no way of determining whether the last block is padded or not).
If | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | if the last block in the ciphertext stream, then is it 6 bytes or 8 bytes of data? If it was supposed to be 8 Bytes then the stream should end as:

8	7	6	5	4	3	2	2	8	8	8	8	8	8	8	8

Block Cipher Modes: Output feedback (OFB)



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

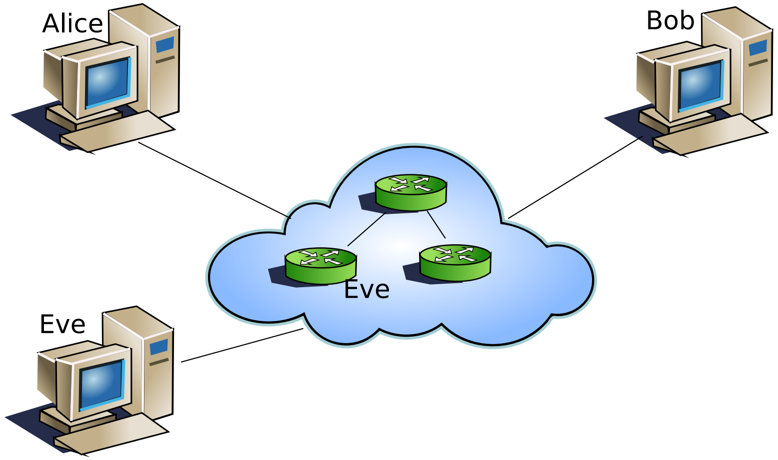
It never ends ... [2013]

When used in a particular way, CBC can be vulnerable to a form of Adaptive Chosen-Plaintext Oracle attack (remember the master key lock example).

SSL (TLS) is a widely used security protocol that uses symmetric keys to secure the session, for example, the session between a web server and a web browser. Some SSL implementations were vulnerable to the adaptive chosen-plaintext attack which was exploited by the *Beast* attack. This meant an attacker could effectively decrypt ciphertext shared between web browser and server.

If used properly, OFB is believed to be OK, however, IETF `draft-sheffer-tls-bcp-00` [Recommendations for Secure Use of TLS and DTLS, Sheffer, Sept 2013] recommend Galois/Counter Mode.
As part of the protocol, the browser and server agree on a block cipher.

Network Security



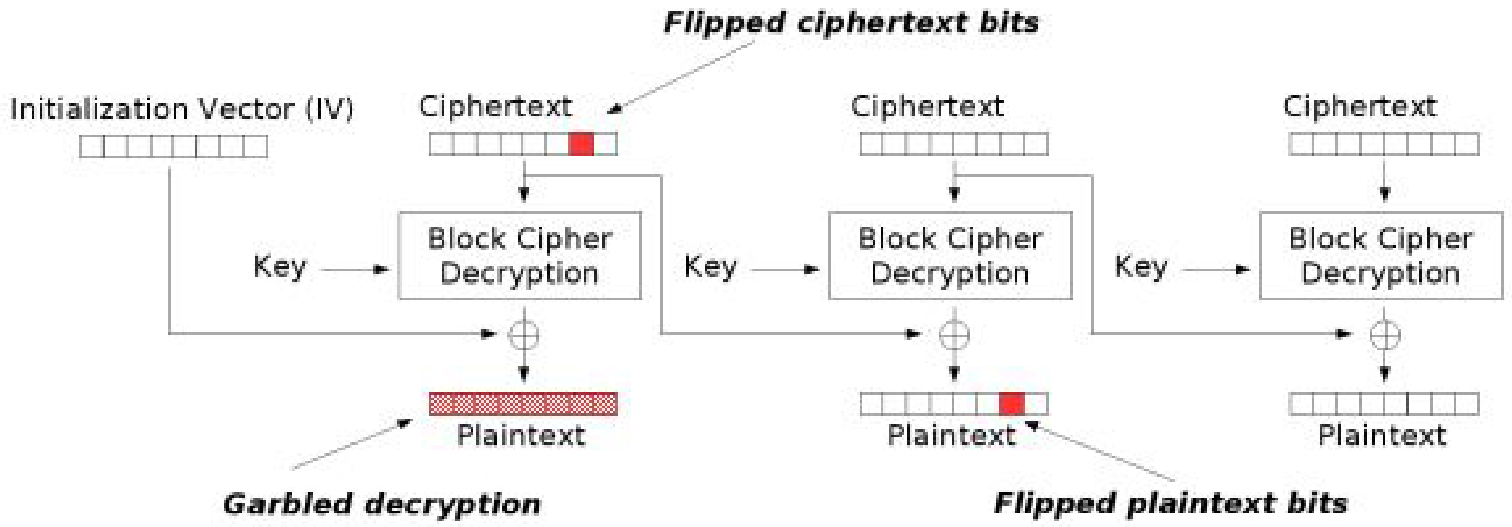
Alice and Bob share a secret symmetric K_{AB} of a cryptographically strong block cipher (for example, 3DES or AES).

$$Alice \rightarrow Bob : E_{CBC}(K_{AB}, M)$$

So long as Alice and Bob choose a good key (not easily guessed) then eavesdropper Eve cannot determine the content of the message M.

Recall message *integrity*: data cannot be altered without detection.

Simply CBC encrypting a message does not provide integrity as the recipient has no way to know whether the message received is valid.



How can a principal detect a garbled decryption?

Remember that the plaintext can be binary data.

Using Plaintext Recognizers do not provide Integrity

Place a recognizer at the end of the plaintext. For example, a block of nulls: if it is not present after decryption then the message has been corrupted.

However, if one corrupts an early ciphertext block, then the recognizer will 'recover' in time:

$$
\begin{array}{rclrcl}
c_0 & = & E(k, p_0 \oplus IV) & p_0 & = & D(k, c_0) \oplus IV \\
c_1 & = & E(k, p_1 \oplus c_0) & ?? & = & D(k, c_1') \oplus c_0 \\
c_2 & = & E(k, p_2 \oplus c_1) & ?? & = & D(k, c_2) \oplus ?? \\
c_3 & = & E(k, \mathsf{NULLS} \oplus c_2) & \mathsf{NULLS} & = & D(k, c_3) \oplus c_2
\end{array}
$$

CBC on its own cannot provide message integrity.

A number of block encryption modes have been designed to provide secrecy and integrity in a single block cipher. Many of these attempts failed, however, some modes proposed for AES are currently *considered* to work, including OCB, XCBC and IACBC.

We will use one-way hash functions to provide message integrity (next section).

Message Authentication Code (MAC) is the last ciphertext block returned when encrypting (CBC) a message M under a secret key.

$$A \rightarrow B : M, \text{MAC}$$

Receiver Bob can check integrity of plaintext M by recomputing MAC and comparing it with received MAC. (often called Message Integrity Code-MIC). Used in ISO-8730

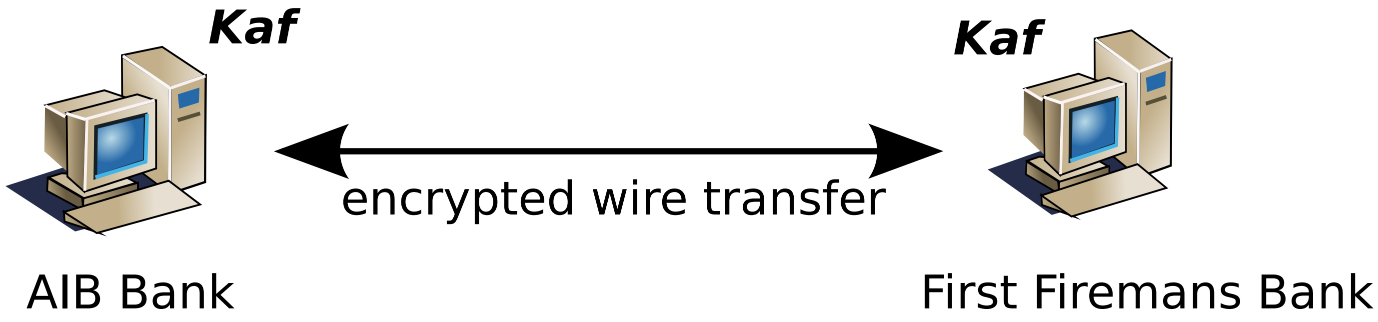
In general, a MAC is a cryptographic checksum that allows one to check the integrity of a message but does not provide secrecy. We need a second cryptographic pass through the message to provide secrecy.

For example, Alice and Bob share secrets K^s_{AB} and K^i_{AB}. Alice computes a MAC of the message using key K^i_{AB} and encrypts the message plus MAC:

$$Alice \rightarrow Bob : E(K^s_{AB}, [M, MAC])$$

A hash function provides a more effective way of achieving secrecy and integrity.

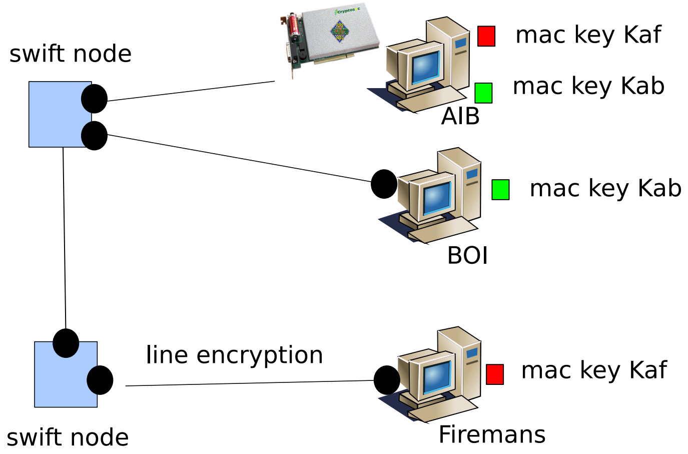
SWIFT(net) provides a financial messaging network for interbank transactions.



> 'To AIB Bank, Ireland. Please pay from our account with you no. 1234567890 the sum of \$ 1000 to John Smith, Patrick Street, Cork who has AIB account no. 301234 4567890123, and notify him that this was for "Invoice Payment 5432346". From First Fireman's Bank of London, UK. Charges to be paid by us.
> Authenticator = 470145D3'

How do banks securely exchange their symmetric key K_{af}?

SWIFT1 is effectively a secure email service



Banks Primary concern is message integrity.

Use MACs for integrity

MAC keys managed end-to-end SWIFT not trusted to manage MAC keys

When a bank sets up a relationship overseas, a senior manager exchanges keys with his/her opposite number, either face to face, or by secure post.

Use two key components to minimize risk of compromise. Key not enabled until both banks confirm key material has been safely received and installed.

Banking Standard ISO8730

SWIFT II uses public-key cryptography for authentication and key exchange (the Bilateral Key Exchange protocol).