Cryptography outline

- open vs. closed system design
- a brief look at historical ciphers
- symmetric ciphers
- hash functions
- Detailed look at DES, construction, modes of operation, vulnerability
- the Advanced Encryption Standard effort
- public key systems
  - a brief look at number theory
  - the importance of primes and how to find them
  - RSA
  - Diffie-Hellman
- key distribution
- basic public key infrastructure

Crypto that is not covered

- Stegonography
- secret sharing
- voting protocols
- zero-knowledge
- lesser-known protocols (e.g. Leighton-Micali)
- formal proofs
- key recover/escrow
- politics

What is cryptology?

• Cryptography
methods to hide and authenticate information

•Cryptanalysis
methods to expose and substitute information

(malicious adversary -- not like that in error-correcting codes)

3

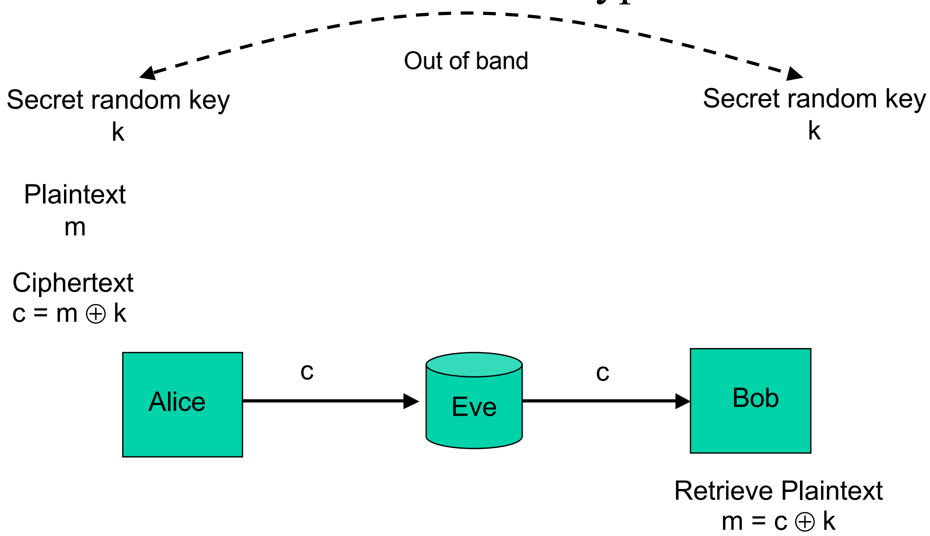
An Important Distinction

Encryption = maintaining information secret/confidential

Authentication = proving and maintaining information integrity

4

One Time Pad Encryption



Perfect Secrecy of OTP Encryption

- Regardless of the plain text, the ciphertext is uniformly random.
- Theorem (Shannon):
  - An adversary may have some a priori information about the plaintext.
  - However, even an infinitely powerful adversary cannot gain any *additional* information about the plaintext after seeing the ciphertext.

Secret Key Cryptography

- OTP achieves Perfect Secrecy but at a tremendous cost—the key must be as long as the total number of bits communicated.
- Secret keys must be a reasonable length.
  - E.g., would like keys of several hundred bits to be used for encrypting several hundred megabytes.
- For short keys, an infinitely powerful adversary *can* learn a great deal about the plaintext:
  - it can perform a trial decryption using every possible key.
- However, realistic adversaries are not likely to spend more than, say, 2^{100} cycles trying to decrypt. That's 32 million years for a terahertz machine.
- Modern cryptography has spent a great deal of time formalizing and quantifying security against time-bounded adversaries.

Secret Key Encryption

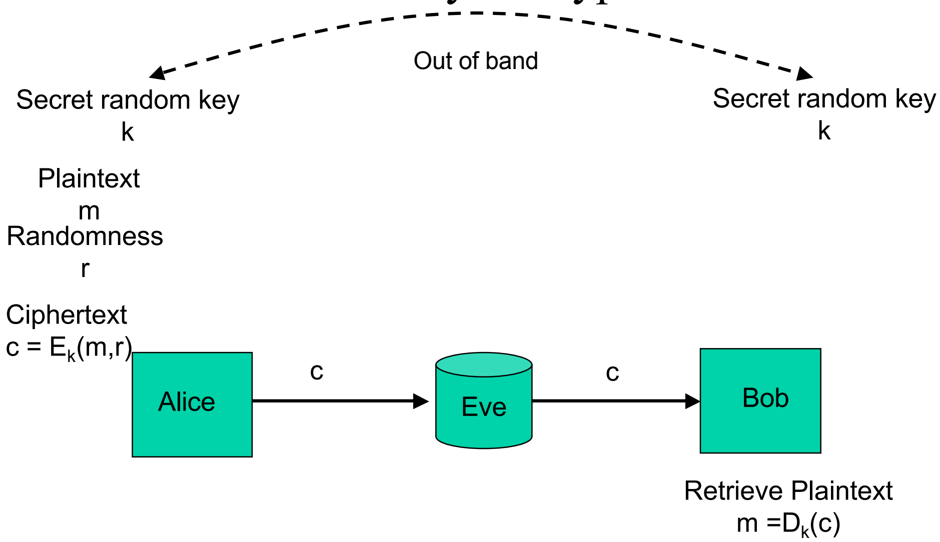
- Alice and Bob have a trust relationship, i.e., they share a secret key, k.
- A secret key encryption scheme is a pair of algorithms:
  1. Encryption algorithm *E* takes a key *k,* a plaintext *m* and a random string *r*, and produces a ciphertext

     $$c = E_k(m,r).$$

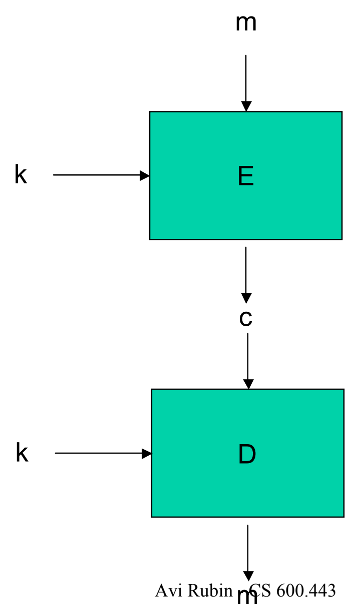
  2. Decryption Algorithm *D* takes a key *k* and a ciphertext *c* and produces a plaintext.
- *E* and *D* must "match," i.e., for all *k, m*, and *r*

$$D_k(E_k(m,r)) = m.$$

Secret Key Encryption



Block Ciphers



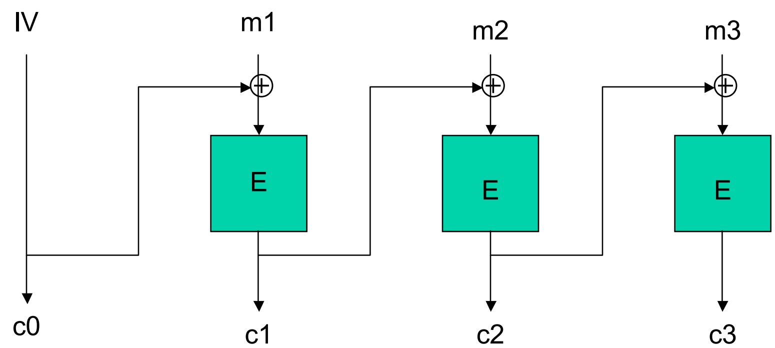
For every k, BC is a Permutation from Strings of length n to Strings of length n.

Examples:

DES, |k| = 56,
|m| = 64

AES, |k| = 128,192, 256;
|m| = 128,192,256

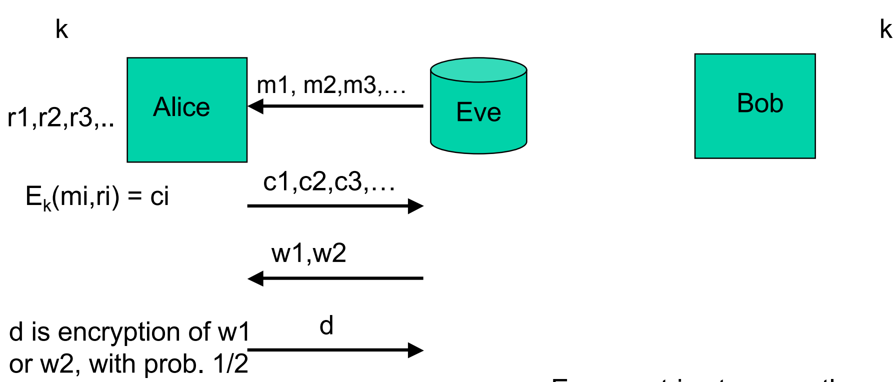
Cipher Block Chaining



$c_i = Ek(m_i \oplus c_{i-1})$;
$c_0 = IV$

Security Against Chosen Plaintext Attack

Security Challenge for Eve: Distinguish between the encryptions of two plaintext of her choosing.



Eve now tries to correctly guess whether d is an encryption of w1 or w2.

Quantified Security

- Quantified Security: An encryption scheme is *(T,ε)-secure* if all adversaries that run in time at most time *T* guess correctly with probability at most ε better than chance (1/2).
- Important point: A secret key encryption scheme must be randomized in order to be secure against a chosen plaintext attack.

Security of OTP and CBC

- Theorem: (infinity,0)-security of OTP
  - No adversary can distinguish encryptions of chosen plaintext with prob better than 1/2 no matter how much time they spend
- Equivalent to Shannon theorem.
- Theorem: CBC with random IV is secure.
  - If the block cipher is difficult to distinguish from a perfect block cipher for a time bounded adversary
  - Then the encryptions of chosen plaintexts are difficult to distinguish for a similarly bounded adversary even after a chosen plaintext attack.

Message Authentication

- Alice and Bob have a trust relationship, i.e., they share a secret key k.
- A message authentication code is two algorithms: one to generate an authentication tag and one to verify it:
  1. MAC takes a key k and a message m and returns a tag t.

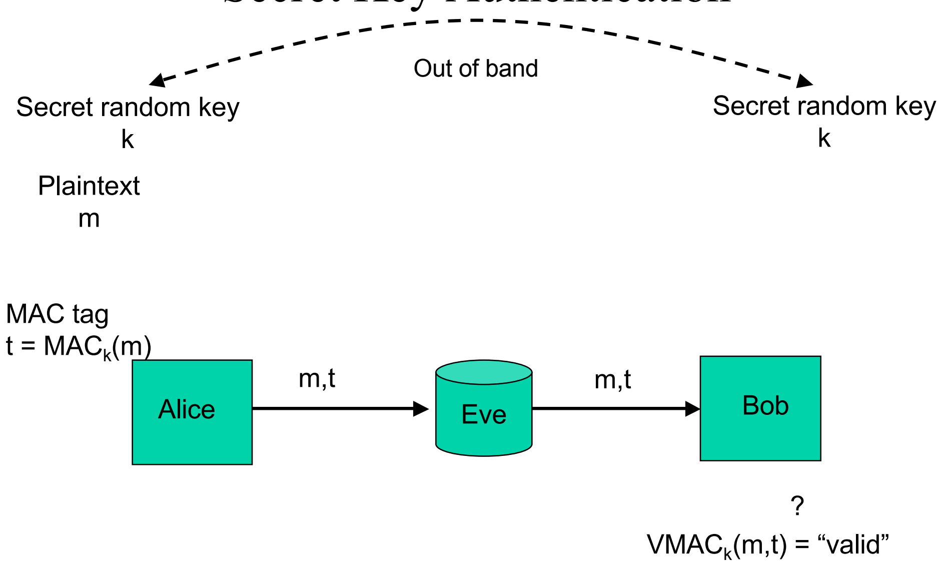
     $$t = MAC_k(m)$$

  2. VMAC takes a key k, the message m and the tag t and says "accept" or "reject."
- MAC and VMAC must "match," i.e., for all k, and m,
  $VMAC_k(m,t)$ = "valid" if $t = MAC_k(m)$, and
  = "invalid" otherwise.

15

Avi Rubin - CS 600.443

Secret Key Authentication

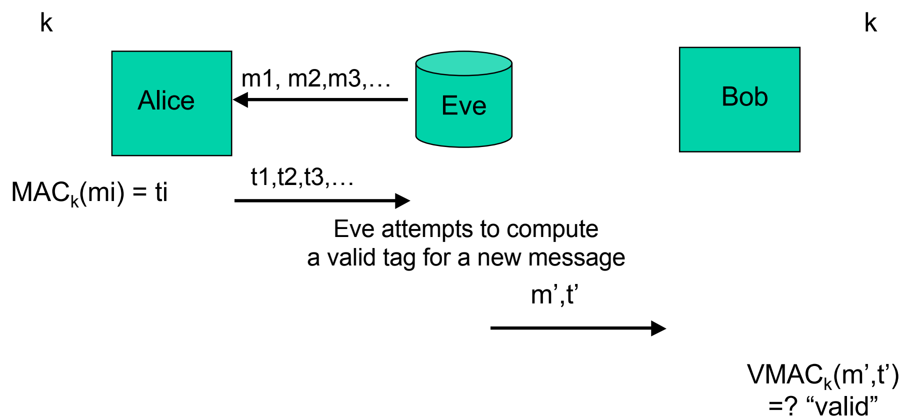


16

Avi Rubin - CS 600.443

Security Against Chosen Plaintext Attack

Security Challenge for Eve: forge a valid message, tag pair



17

MAC Security

- A MAC scheme is secure if even adversaries with significant computing resources have only a small chance of computing a forgery.
- Hence, Bob can trust that messages with a valid tags were sent by Alice.
- Security can be quantified in terms of time devoted by adversary and chance of successful forgery.
- Lots of constructions of MACs. About an order of magnitude faster than encryption.

18

Public Key Signatures

- A public (asymmetric) key signature scheme has three components:
  1. A key generator G.
     - Input: length n.
     - Output: Secret signing key, sk, and public verification key, vk of length n.
  2. A signature algorithm S—uses the secret signing key
     - Input the signing key sk, & a message m
     - Output: a signature $s = S_{sk}(m,r)$.
  3. A verification algorithm V—uses the public verification key
     - Input: the verification key vk, a message and a purported signature s
     - Output: "valid" or "invalid."
- S and V must be well "matched," i.e.,

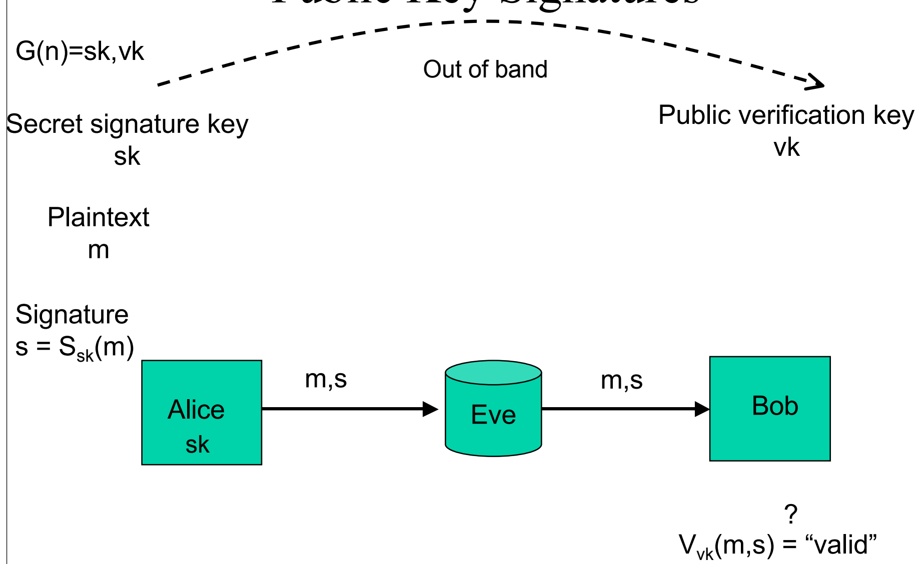
  $V_{vk}(m,s)$ = "valid" if $s = S_{sk}(m,r)$, and

  = "invalid" otherwise.

19

Avi Rubin - CS 600.443

Public Key Signatures



20

Avi Rubin - CS 600.443

Public Key Signatures

- Definition nearly the same as that of a MAC except that the signing and verification keys are different and the latter may be public.
- Security against choose message attack defined very similarly to previous definition of message authentication codes.
- Constructions based on the famous RSA problem.
- "Provable" schemes: Show that if all adversaries with considerable resources have only a very small chance of inverting RSA then all adversaries with similar resources have only a very small chance of forging a signature.

Example: RSA

- Key Generation for 1024 bit RSA:
  - Generate two 512 bit primes p and q
  - Set N := p*q
  - Let R := be the set of all integers < N that do not have p or q as a prime factor.
  - One half of the trick: given p and q, it's easy to compute two integers

    d and e

    with the following special property:

    $(x^d)^e \bmod N = x,$ for x in R.

  - Set public verification key to (N,e)
  - Set private signing key to (N,d)

RSA, continued

- Given m in R, & signing key (N,d), the RSA signature s is computed as

$$s = m^d \bmod N$$

- Given a message and its signature (m,s), & verification key (N,e), an RSA signature is (publicly) verified by checking:

$$s^e \bmod N =? \ m$$

- If s is a valid signature then s will pass the test:

$$s^e \bmod N = (m^d)^e \bmod N = m$$

- The second half of the trick: Given the public key (N,e) its not feasible to compute the signing key (N,d).
  - So, adversary can't sign messages.

How Many Prime Numbers Are There Anyway?

- The number of Prime numbers not exceeding "N" is approximately "N/log(N)."
- If "K" is a randomly chosen odd number,
  - Prob("K is prime") = (2 / log(K)).
- if K is 512 bits, Prob("K is prime") = 2/177
- it takes about 89 tests to find one prime

Primes

- RSA relies on prime numbers
- Where do we get prime numbers?
- Could a list of primes be published?
- Are there enough primes in the world to make exhaustive search hard?
- How do you know if a number is really prime?

How to Find Big Prime Numbers?

- How to verify/test if a number K is prime?
- How many K's do we have to test before we can hit a prime?
  - try about (log (N) / 2) different random odd numbers.

Fermat's Theorem

- Given, a $0<a<n$ and n is prime, then:

  $a^{(n-1)} \bmod n = 1$

  Examples
  - $a = 4, n = 5, 4^{(5-1)} = 4^4 = 256 = (255+1) \bmod 5 = 1$
  - $a = 2, n = 5, 2^{(5-1)} = 2^4 = 16 = (15+1) \bmod 5 = 1$

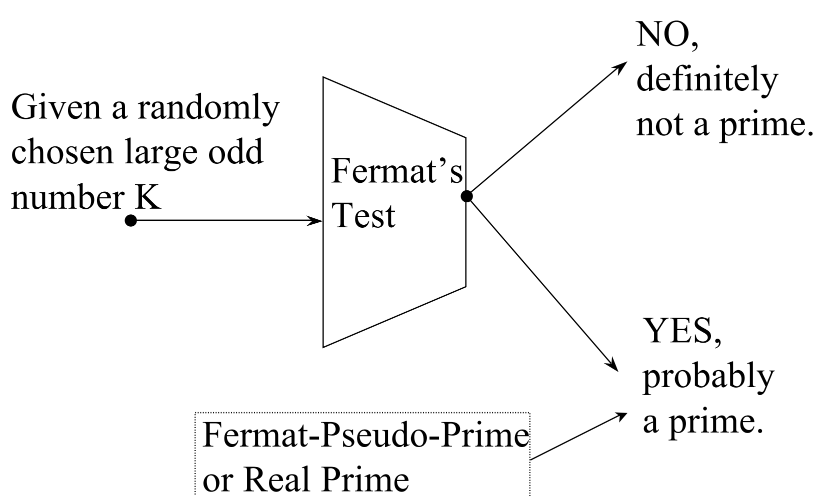
Fermat's Test

- Given n, randomly choose an *a*; compute $a^{(n-1)}$
- If $a^{(n-1)} \bmod n \neq 1$
  - then n is NOT a prime so select another n;
- If $a^{(n-1)} \bmod n = 1$, for n of 100 digits, there is 1 in 10^{13} chance that n is still NOT a prime
  - n is a Fermat-Pseudo-Prime
  - Try test for many different values of a

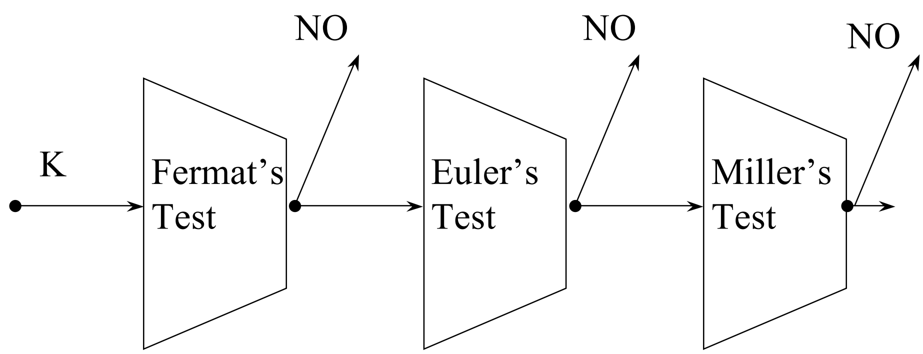
How Likely is Fermat Pseudo Prime?

- If a = 2
- For numbers under 10^{10}, (~100 bits),
  - there 455,052,512 real primes,
  - 14,888   Fermat-Pseudo-Primes.
- Probability = 0.005%.
  - So Fermat pseudo primes are hard to find
  - A lot of processing to find prime number in general
  - So, key generation is slow
  - Precomputation of primes is possible

Probabilistic Primality Testing



Fermat, Euler, Miller Tests



Some More Rigorous Tests

- Euler and Miller Tests are much more rigorrous than the Fermat Test.
- There are:
  - very few Fermat-Pseudo-Primes
  - far fewer Euler-Pseudo-Primes
  - even far fewer Miller-Pseudo-Primes
- In practice, if a K can pass all three tests, with extremely high probability, it is a real prime

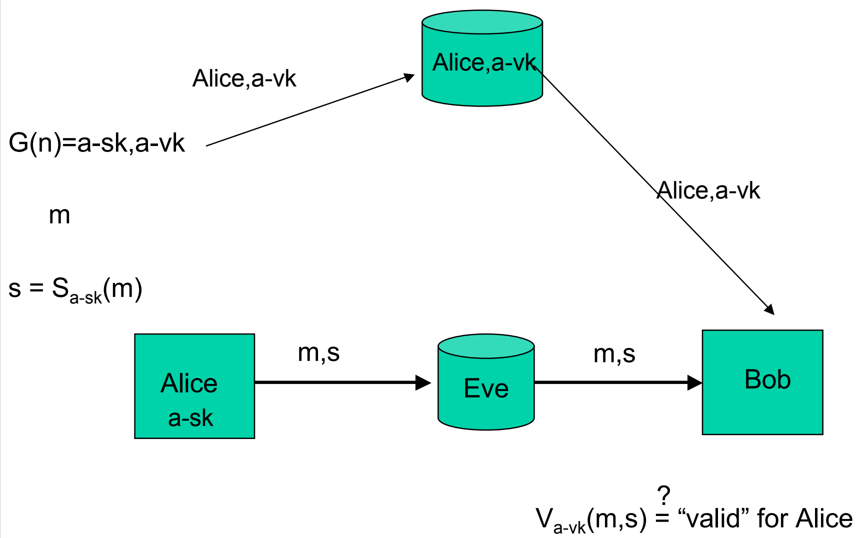
So You Wonder...

- Why not make certain that p and q are primes?
  - For n of 256 bits, testing all $x < 2^{128}$ on a 4 gigaflops machine will take about $3x10^{19}$ years
- What's the problem if they are not really primes?
  - The algorithm fails - the reverse transformation may get the wrong "thing"
  - Cracking private component may no longer be as hard

Public Key Signatures

- Main idea of Public Key cryptography: Alice can use public information to send authentic messages to Bob.
  - That is, Alice and Bob don't need to establish a secret key or a trust relationship ahead of time.
- Attempt 1:
  - Alice runs the key generator to get (a-sk, a-vk).
  - She puts her name and the verification key (Alice, a-vk) in a public key "telephone book" and keeps the signature key a-sk secret.
  - Now, Alice can send a signed message (m,s) to anyone, say, Bob.
  - Bob can then retrieve the verification key a-vk from the directory and use it to verify that the message came from Alice, that is, s could have only been generated from m using the secret key associated with a-vk.

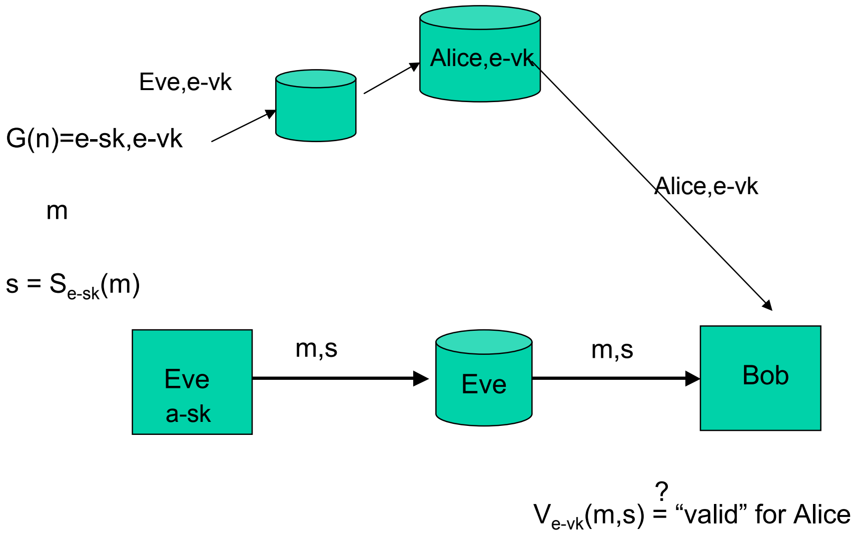
Public Key Signatures



Public Key Signatures, cont’d

- Not quite good enough. What if Eve generates (e-sk, e-vk) and places (Alice, e-vk) in the directory. Then everyone will think that messages signed by Eve were signed by Alice.

Public Key Signatures



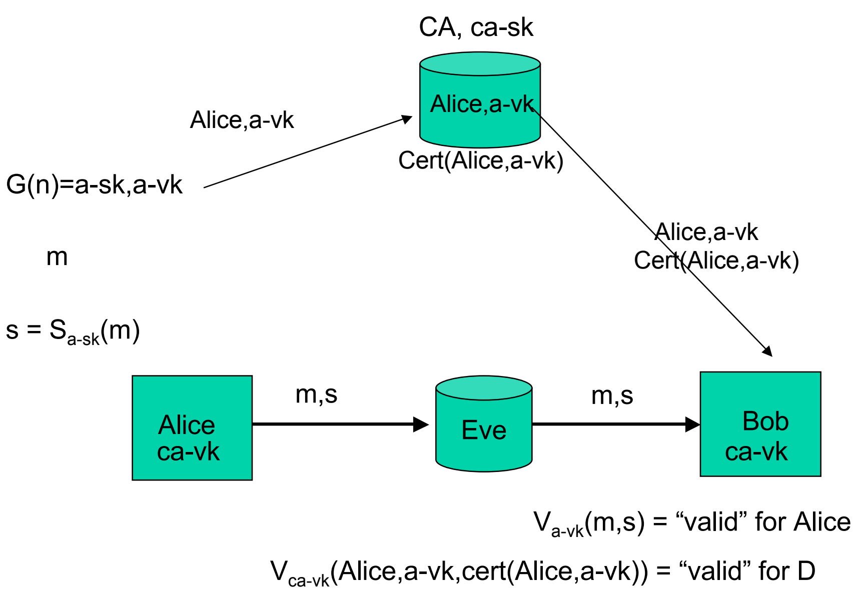
Certificate Authorities

- Attempt 2: First establish trust with the directory, D.
  - Suppose that both Alice and Bob have the verification key d-vk of the directory D, and through some out of band means they trust that d-vk is indeed the verification key of D.
  - Moreover, suppose that the directory has verified Alice's indentification info and her verification key.
  - Suppose the directory computes the signature of: "D certifies Alice's verification key is a-vk" and calls it D-cert-of-Alice, short for Alice's public key certificate.
  - The directory entry for Alice is now: {Alice, a-vk, D-cert-of-Alice}.

Certificate Authority

- Attempt 2, con't:
  - Now Bob's verification procedure of a signed message purportedly from Alice takes three steps:
    1. Retrieve Alice's public key and Alice's public key certificate from the directory
    2. Run V on Alice's public key certificate using the directory's verification key: $V_{d\text{-}vk}$(D-cert-of-Alice).
       - Since Bob trusts that d-vk is really the verification key of D and that D has verified the "binding" of Alice's indentification info to her key, if the certificate is valid, then Dan trusts that the verification key a-vk in the certificate is really Alice's.
    3. Run V on Alice's message and signature using a-vk.
       - If it is valid, then Bob trusts that the message came from Alice.

Public Key Signatures



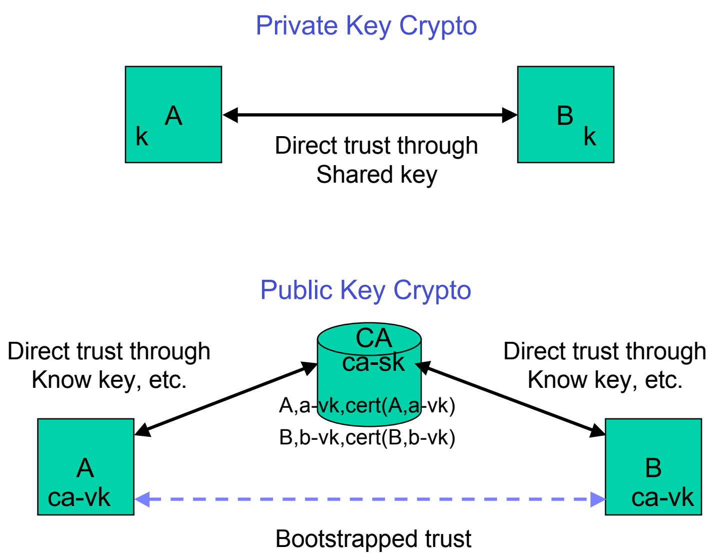
Public Key Infrastructure

- Main Idea of Public Key Cryptography-II:
  - Suppose two parties A & B have no prior trust relationship
  - Suppose both A & B have a trust relationship with C.
  - Then A & B can bootstrap a trust relationship using their mutual trust of C.
- Big difficulty with Public Key cryptography: Revocation
  - A CA may want to revoke a key for a large number of reasons: computer penetration, employee leaves,…
  - Revocation is a fact of life.
  - Difficult to achieve revocation which is: efficient, accurate, simple, and scalable.

41

Avi Rubin - CS 600.443

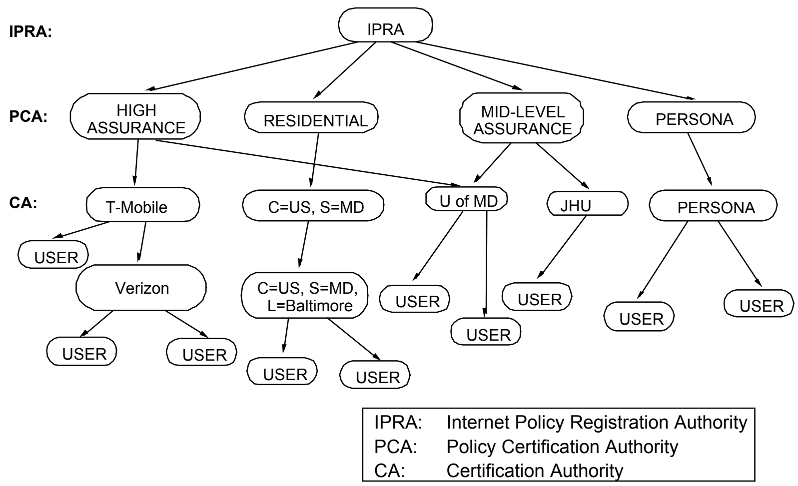
Trust Relationships



42

Avi Rubin - CS 600.443

Certification Hierarchy



43

Avi Rubin - CS 600.443

Policy Decisions for Certification

- CA issues
  - Number, names, and locations of CAs
  - Protection of the CA's private key
  - Identification of principals
  - Naming: Roles, titles, and authorizations
  - Lifetime of certificates
  - Generation, use, and protection of principals' keys
  - Frequency of CRL update
- Other issues
  - Requirements for display and verification
  - Record retention
  - Storage of encrypted data
  - Emergency access to keys

44

Avi Rubin - CS 600.443

Variations on Strict Hierarchies

- Cross certification among CAs
  - More efficient look up
  - Allows continued operation if higher point in common is compromised
  - Becomes unwieldy when over used
- Bottom up "islands of trust"
  - Intended to solve start up problems
  - May lead to policy divergence later
- Web of trust
  - Users exchange keys like business cards
  - Multiple paths is the norm
  - Judgments about transitivity left to the user

45

Avi Rubin - CS 600.443

Public Key Encryption

- A public key encryption scheme has three components:
  1. A key generator G.
     - Input: length n.
     - Output: a public encryption key, ek, and a private decryption key dk of length n.
  2. An encryption algorithm E—uses the public encryption key
     - Input: the encryption key ek, random bits r, & a message m,
     - Output: a ciphertext $c = E_{ek}(m,r)$.
  3. A decryption algorithm D—uses the private decryption key
     - Input: the decryption key dk, and a ciphertext c.
     - Output: a message $m = D_{dk}(c)$
- E and D must be well "matched," i.e., for all r and m,

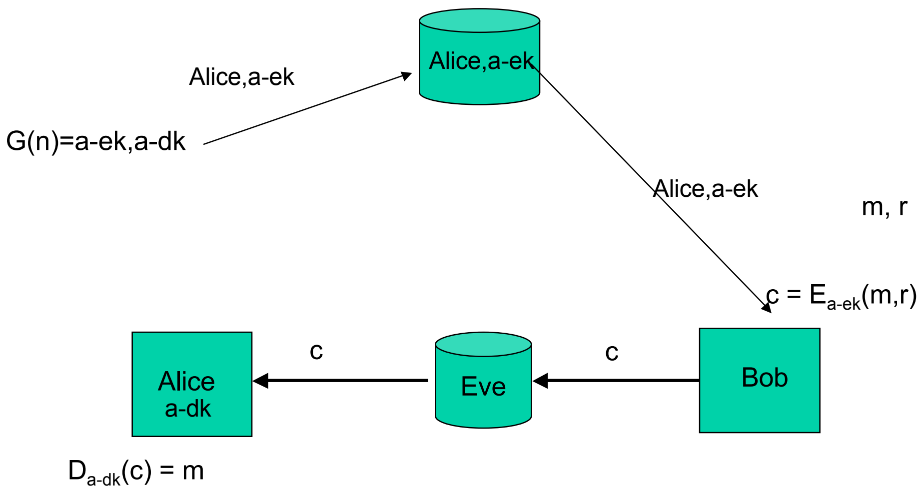
$$D_{dk}(E_{ek}(m,r)) = m.$$

- Definition of security against chosen plaintext attack similar to the definition for secret key encryption.

46

Avi Rubin - CS 600.443

Public Key Encryption



47

Combining Public and Private Key Schemes

- Public Key schemes are *much* more time consuming than private key schemes.
- Typically, if two parties do not share a secret key, they first engage in a “secret key agreement protocol” based on public key encryption and signatures.
- Once they have established a shared secret key, they use efficient secret key encryption and MAC schemes to protect their communication.

48

A Simple taxonomy

	Encryption	Authentication
Symmetric key	Block ciphers: DES, AES Stream ciphers: RC4	HMAC MMH MAC
Public key	RSA encryption El Gamal encryption	RSA DSA

Pseudo-random number generators

Problems:

- getting many truly random bits is slow
- getting many shared truly random bits is more awkward
- getting "good randomness" is important for many crypto algorithms

Solution:

- theory: pseudo-random strings that are "polynomial time indistinguishable" from truly random strings
- practice: use DES, hash functions generate bits from a random *seed* (FIPS 186)

Digital Fingerprints: One-Way Hash Functions

* Cryptographically “compress” any message, M, to a fixed sized string, H(M).
* Design goal is to make it computationally infeasible to find <u>any</u> M_1 and M_2 with $H(M_1) = H(M_2)$.
* What good are one-way functions?
  1. Encrypting passwords UNIX, OPIE
  2. Constructing digital signatures
  3. Message integrity and authentication
  3. Part of many other cryptographic applications:
     – Pseudo-random generators
     – Identification protocols
     – Coin flipping by telephone
     – Digital timestamping

OPIE (a.k.a S/KEY)

- Initialization - on secure machine
  – user enters password, pw and n
  – User computes:
    pw_n = f(f(f(...f(pw))))...) n times
    where f is a one-way function
  – User sends pw_n to server
  – Server stores pw_n

Opieinit (cont.)

Client #1		*Server*
pw_0 = user password		
$pw_1 = f(pw_0)$		
$pw_2 = f(pw_1)$		
$pw_3 = f(pw_2)$		
$pw_4 = f(pw_3)$		
...		
$pw_n = f(pw_{n-1})$	----->	client #1, $pw_n = f(pw_{n-1})$

53

OPIE (cont.)

- To authenticate
  - Server knows $f^n(pw)$
  - Client known pw

  Client -> Server : "I wish to authenticate"

  Server -> Client : n

  Client computes $f^{n-1}(pw)$

  Client -> Server : $f^{n-1}(pw)$

  Server computes $f(f^{n-1}(pw))$

54

Example OPIE one-time passwords

464: DAN MAP FAIR CLAN HOVE BOO
465: TOP JAM CULT MOLT LAWN SEEN
466: SLID RODE JIG SLUG HUE COIN
467: SWAG IT AMES ELI WAST TIP
468: TIP SMOG EGAN MAP VIEW AJAR
469: EEL STAG SKIT AID DONE SLY
470: SKI APT BAND KIND BAD AD
471: BOB FREY HIDE FUSS GARY LAP
472: FIRE HUCK MIND DUE REEL KUDO
473: AGO AWRY WIT HAY BULK RAW
474: TIM KNOT KEY HASH FUM PAP
475: LYNN FIVE LILY JUG FARM AVON
476: COL COOT COLD FOOL NAGY MESH
477: NOON CHEN NAIL GAB SEEM GALA

Birthday attacks

- Alice prepares two version of a contract
  - one very favorable to Bob - contract 1
  - the other would bankrupt Bob - contract 2
- Alice makes subtle changes to contract
  - e.g. replace a space with space-backspace-space characters
  - by making or not making change on 32 lines, 2^{32} different docs.
- Alice compares hash documents for both docs with all changes
  - if hash output 64 bits, should find a match using 2^{32} different docs
- Alice gets Bob to sign contract 1 of contract for which she has a contract 2 collision
- Alice can convince a judge that Bob signed contract 2.

Lessons

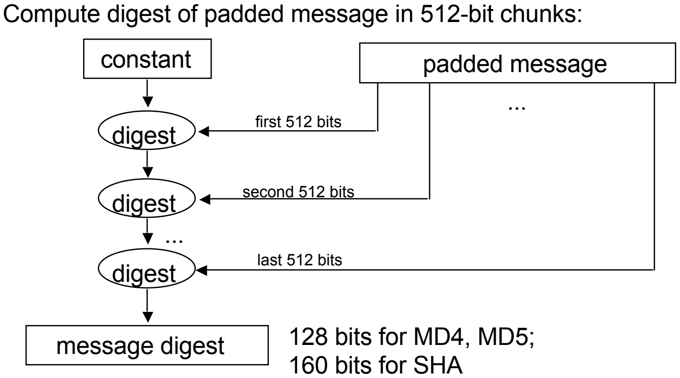
- Use has function with long output 160 bits would require 2^{80} documents
- Always make some cosmetic change to a document before signing
- Compress before signing
  - eliminates redundancy
- Hash message, append hash to message, hash again.
  - hash value is the two hash results concatenated together
  - this method never proven secure or insecure

Structure of MD4, MD5, and SHA

1. Pad message to a multiple of 512 bits:

original message	1000 ... 000	64-bit original length

2. Compute digest of padded message in 512-bit chunks:



Data Encryption Standard (DES)
(symmetric key)

$\text{Enc}_k(M)$ =
wild permutation, XOR's of M, S-boxes, and k

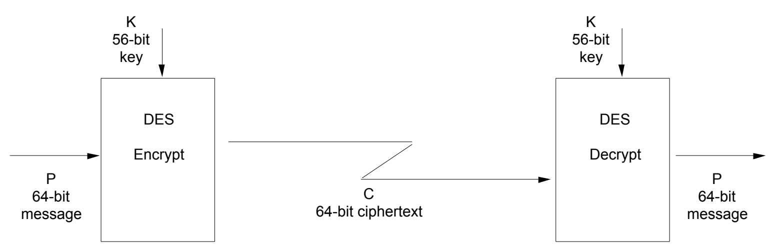
16 "rounds," 64-bit block input and output
not clean and concise (like RSA and one-time pad)

Standard for encryption of unclassified data since 1977

56 bits yield valid concerns about vulnerability to "exhaustive key search"

59

DES—The Data Encryption Standard



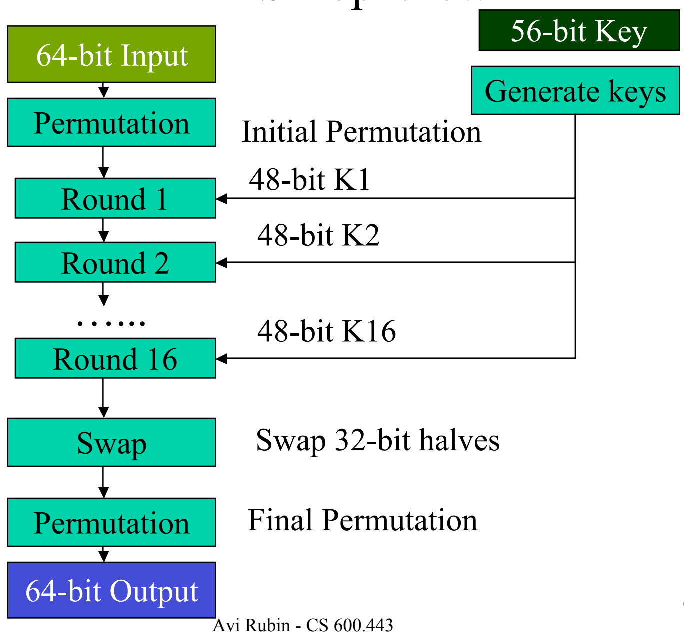
Benefits:
- Publicly known
- Reasonably fast
- FIPS specification
- Widely used
- Extensively analyzed
- No major defects found

Drawbacks:
- Key size is marginal.
- Was not designed for software.

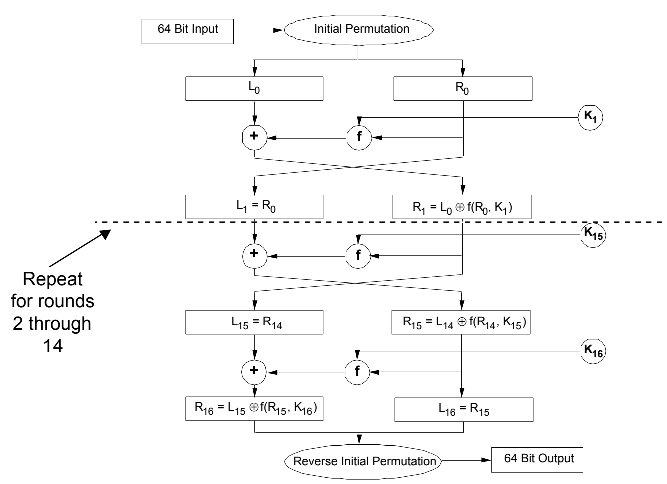
Implementation Issues:
- Blocks are independent.
- Key distribution needed.

60

DES Top View

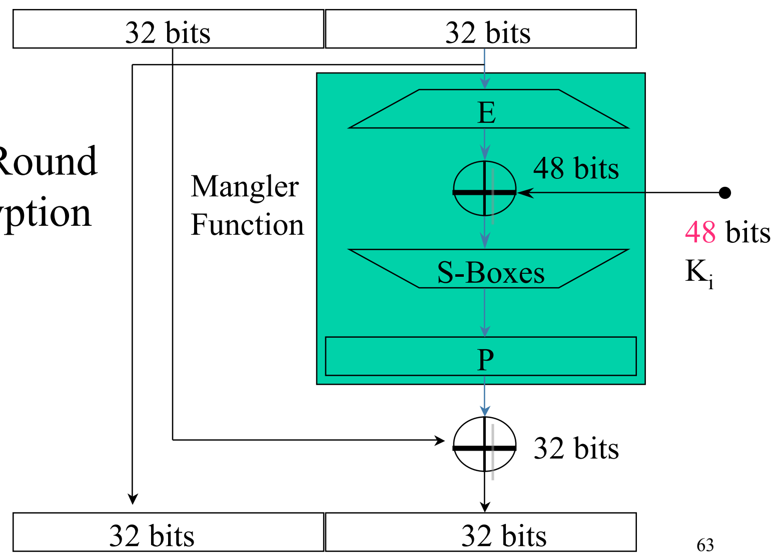


DES — 16 Round Structure

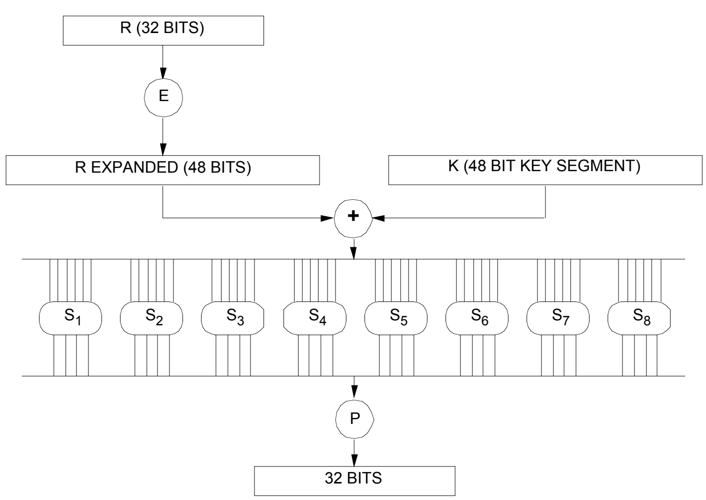


Cipher Iterative Action

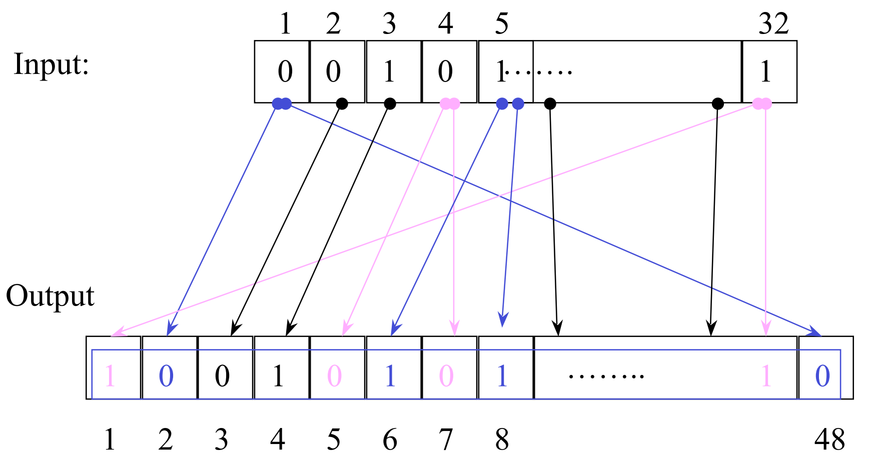
One Round Encryption



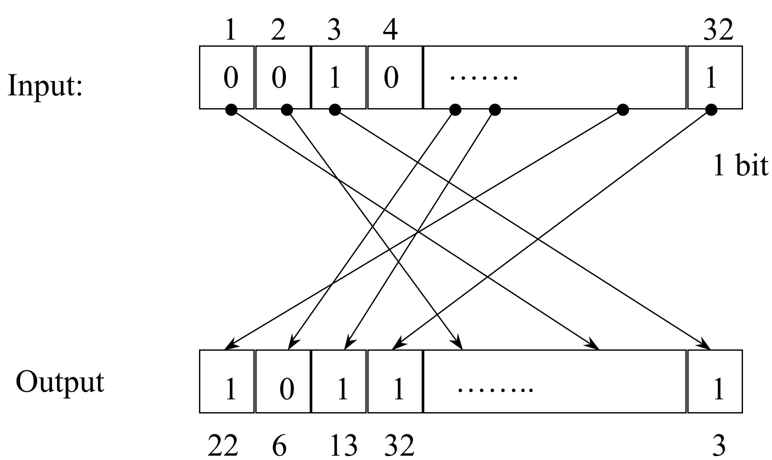
A Single Round of DES



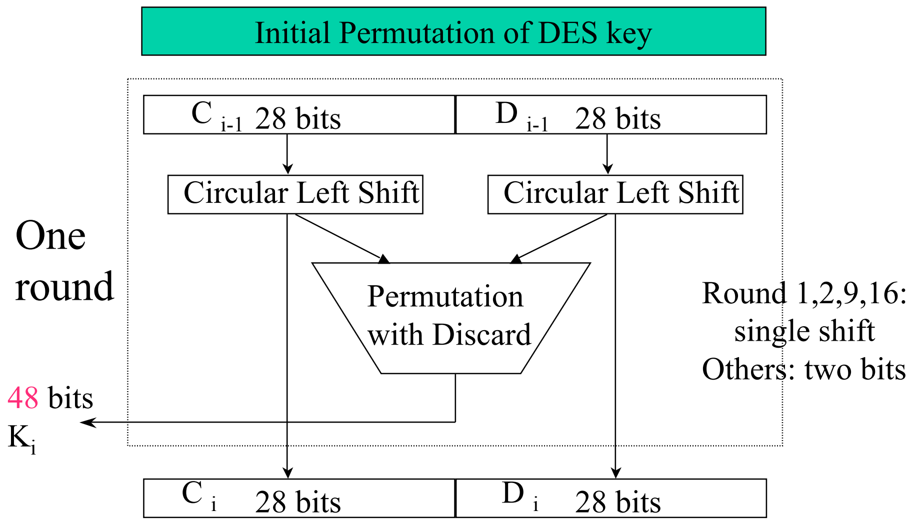
Bits Expansion (32-to-48)



Bit Permutation (32-to-32)



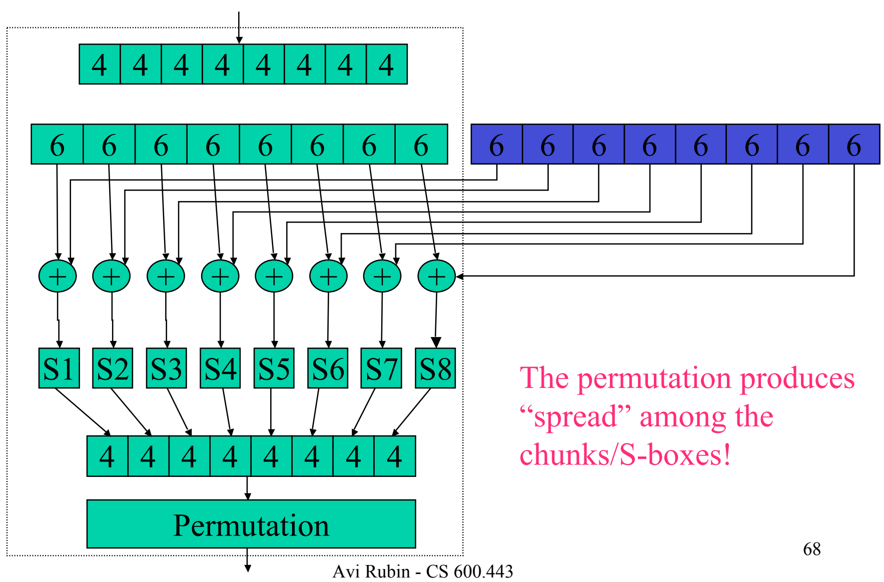
Per-Round Key Generation



67

Avi Rubin - CS 600.443

Mangler Function



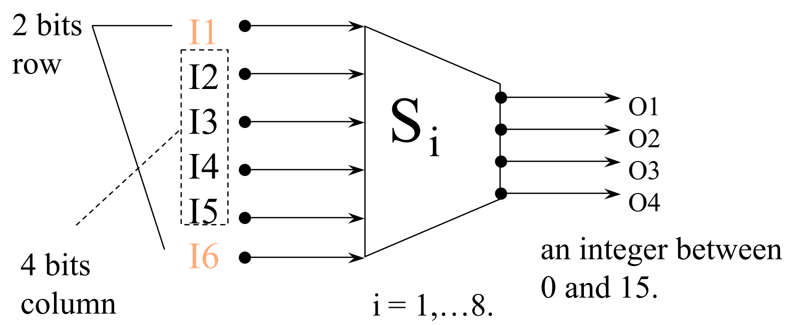
The permutation produces "spread" among the chunks/S-boxes!

68

Avi Rubin - CS 600.443

S-Box (Substitute and Shrink)

- 48 bits ==> 32 bits. (8*6 ==> 8 *4)
- 2 bits used to select amongst 4 permutations for the rest of the 4-bit quantity



69

Avi Rubin - CS 600.443

S1

Each row and column contain different numbers.

	0	1	2	3	4	5	6	7	8	9…. 15
0	14	4	13	1	2	15	11	8	3	
1	0	15	7	4	14	2	13	1	10	
2	4	1	14	8	13	6	2	11	15	
3	15	12	8	2	4	9	1	7	5	

Example: input: 100110 output: 8 = 1000

70

Avi Rubin - CS 600.443

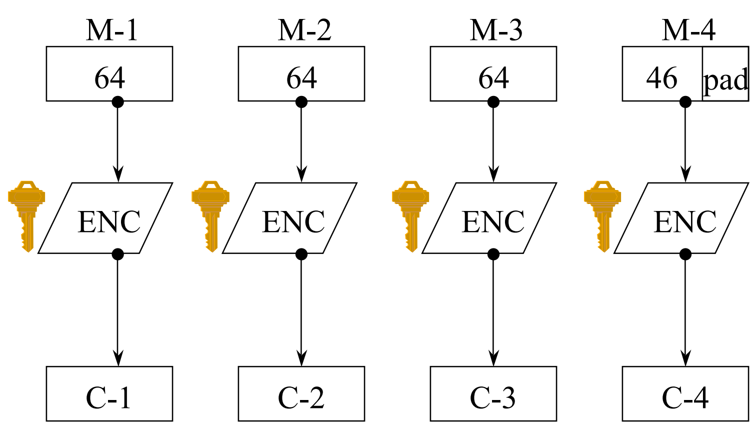
S/Box criteria

- Mostly unpublished
- Several revealed
  - No S/Box is a linear affine function of its inputs (no linear system of equations to express output bits in terms of input bits)
  - change one input bit => change two output bits (maximize diffusion)
  - minimize difference between number of 1s and 0s. E.g. hold one bit constant and change other 5 bits, and output has similar number of 1s and 0s.
- Almost any change results in insecure cipher

71

Avi Rubin - CS 600.443

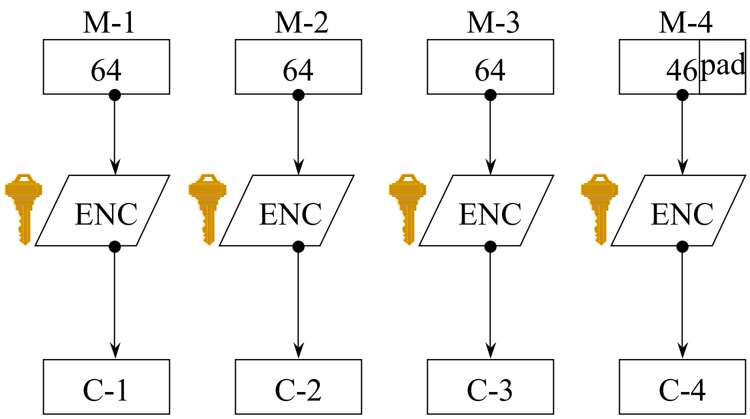
Electronic Code Book (ECB)



72

Avi Rubin - CS 600.443

ECB problem #1



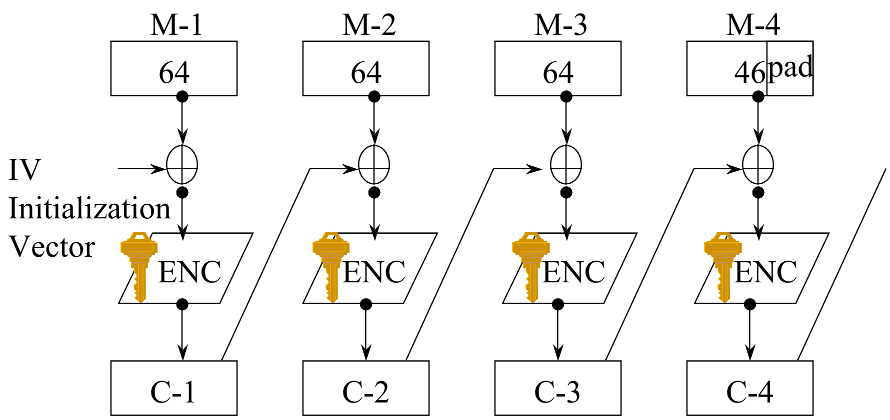
(M-1 == M-3) => will (C-1 == C-3)

ECB preserves the patterns in plaintext too well!

ECB Problem #2

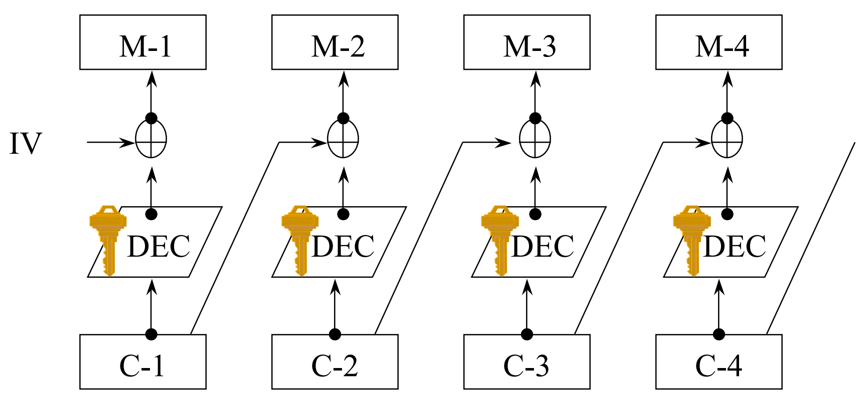
- Lack the basic protection against integrity attacks on the ciphertext at message level (I.e., multiple cipher blocks)
- Without additional integrity protection
  - cipher block substitution and rearrangement attacks
  - fabrication of information

Cipher Block Chaining (CBC)



(M-1 == M-3) very unlikely leads to (C-1 == C-3)

CBC Decryption



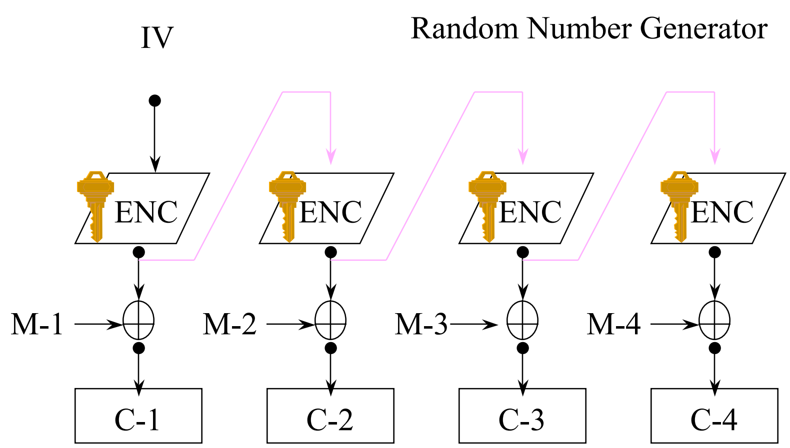
CBC - properties

- Patterns in plaintext are concealed
- an error is only propagated one block
- can do random access decryption
  - one block look backwards required
- can use last block as integrity check on message
- need to keep track of length and padding

Avi Rubin - CS 600.443

77

Output Feedback Mode (OFB)



Avi Rubin - CS 600.443

78

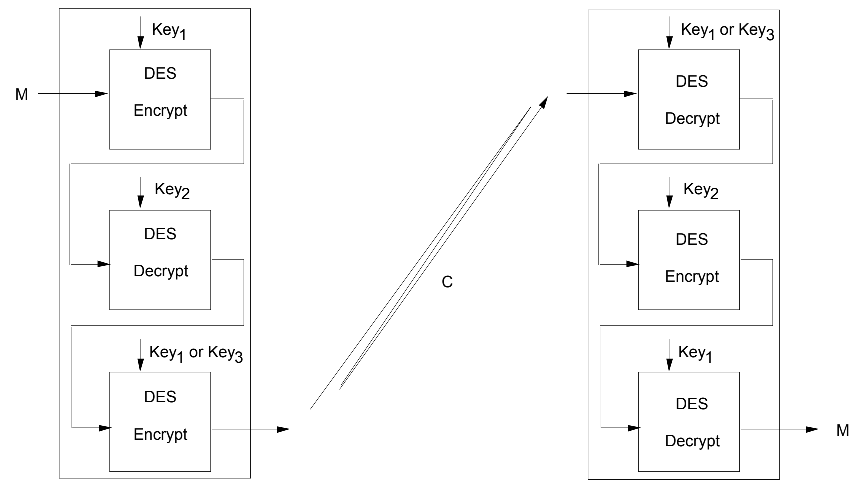
OFB Properties

- Allow pre-computing of pseudorandom stream (One-Time Pad); Xor can be implemented very efficiently
- No error propagation problem as in CBC
- Allow in-time encrypt/decrypt due to bit-wise computation (versus the fixed block)
- Loss of sync implies loss of data

Attacks on DES

- Exhaustive search
  - look for 2^{55} keys on average
  - specialized hardware
  - Wiener attack: $1 million and three months per key
  - Gilmore's machine $250,000, 3 days on average
- Linear cryptanalyses 2^{47} known plaintext pairs
- Differential cryptanalyses 2^{43} chosen plaintext pairs
- Double DES no more secure than single DES
  - meet in the middle attack

Triple DES (3-DES)



81

Avi Rubin - CS 600.443

DESX

- Uses 3 keys
- Requires one round of DES
- Effective key length (exhaustive search cost)
  - believed to be around 112

DESX: ciphertext = k1 xor EncDES_{k2}(message xor k3)

82

Avi Rubin - CS 600.443

RC4

RC4 is a stream cipher with a key schedule algorithm:

Initialization: (state array *S*)

```
For i=0…N-1
  S[I]=i
J=0
```

Scrambling:

```
For i=0…N-1
  j = j+S[i]+K[i mod m],  (key has m bytes)
  Swap(S[i],S[j])
```


RC4

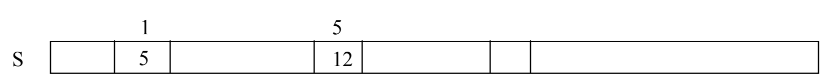
- After key setup, comes PRG (independent of key):

Initialization:

```
i=0
J=0
```

Generation loop:

```
i=i+1
j=j+S[i]
Swap(S[i],S[j])
Output z = S[S[i] + S[j]]
```



RC4

- After key setup, comes PRG (independent of key):

Initialization:

```
i=0
J=0
```

Generation loop:

```
i=i+1
j=j+S[i]
Swap(S[i],S[j])
Output z = S[S[i] + S[j]]
```

		1		5		17	
S		12		5		3	

AES

DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
[Docket No. 970725180-8168-02]
RIN No. 0693-ZA16

REQUEST FOR COMMENTS ON CANDIDATE ALGORITHMS FOR THE ADVANCED ENCRYPTION STANDARD (AES)

- Evaluation criteria
  - Security
  - Cost
    - no licensing (worldwide, non-exclusive, royalty-free basis)
    - Computational efficiency
    - Memory requirements

AES (cont)

- Evaluation (cont).
  - Algorithm and Implementation Characteristics
    - Flexibility (key size, block size, time/memory tradeoffs)
    - Hardware and software suitability
    - Simplicity of design
- Timetable
  - On August 20, 1998, at the First AES Candidate Conference, NIST announced the 15 AES candidates for Round 1 evaluation
  - Round 1, August 20, 1998 - April 15, 1999
  - Second AES Candidate Conference was held on March 22-23, 1999, in Rome, Italy

87

AES (cont.)

- Timetable (cont.)
  - Summer, 1999 finalists announced
  - 3rd AES conference April 10-14 in NYC.
- Candidates
  - must submit code
  - must have no intellectual property problems
  - must submit full specification

August 9, 1999 - NIST Announces the AES Finalist Candidates for Round 2:

MARS, RC6TM, Rijndael,Serpent, and Twofish

88

Avi Rubin - CS 600.443

AES (cont.)

- Meanwhile
  - tons of Crypto & Eurocrypt papers
  - NIST performed statistical and efficiency testing on candidates
  - several candidates losing chance
- Example test
  - Time to encrypt 1 megabyte
  - Time to decrypt 1 megabyte
  - Time to generate 1000 key pairs (enc/dec)
  - key setup time
  - cycle round counting

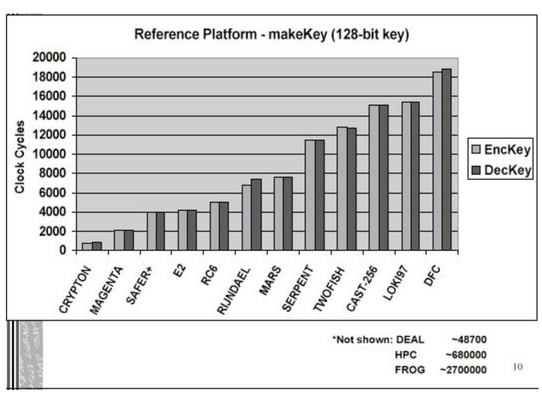
Candidates

- CAST-256 Entrust Technologies, Inc. (represented by Carlisle Adams)
- CRYPTON Future Systems, Inc. (represented by Chae Hoon Lim)
- DEAL Richard Outerbridge, Lars Knudsen
- DFC CNRS - Centre National pour la Recherche Scientifique - Ecole Normale Superieure (represented by Serge Vaudenay)
- E2 NTT - Nippon Telegraph and Telephone Corporation (represented by Masayuki Kanda)
- FROG TecApro Internacional S.A. (represented by Dianelos Georgoudis)
- LOKI97 Lawrie Brown, Josef Pieprzyk, Jennifer Seberry
- HPC Rich Schroeppe
- MAGENTA Deutsche Telekom AG (represented by Dr. Klaus Huber)
- MARS IBM (represented by Nevenko Zunic)
- RC6 RSA Laboratories (represented by Matthew Robshaw)
- RIJNDAEL Joan Daemen, Vincent Rijmen
- SAFER+ Cylink Corporation (represented by Charles Williams)
- SERPENT Ross Anderson, Eli Biham, Lars Knudsen
- TWOFISH Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

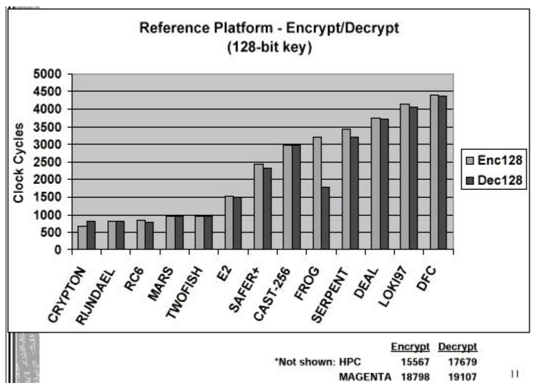
AES (cont.)

- Performance
  - benchmarks on all sorts of hardware and O/S
  - Specify compilers and options

How fast is makeKey?



How fast to en/de crypt?

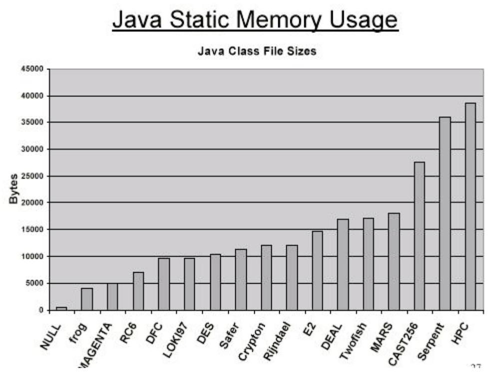


Side by side comparison

“Reference” Configuration Results

Algorithm	setKey(enc)	setKey(dec)	Encrypt	Decrypt
CAST-256	15028	15028	2971	2983
CRYPTON	720	805	669	803
DEAL	48762	48776	3748	3729
DFC	18521	18804	4418	4359
E2	4197	4162	1523	1509
FROG	2686986	2707347	3208	1784
HPC	675955	680980	15567	17679
LOKI97	15335	15347	4156	4054
MAGENTA	2112	2108	18798	19107
MARS	7622	7621	964	945
RC6	5015	5014	845	786
RIJNDAEL	6787	7467	809	832
SAFER+	4026	4023	2420	2318
SERPENT	11398	11400	3424	3217
TWOFISH	12799	12677	973	965

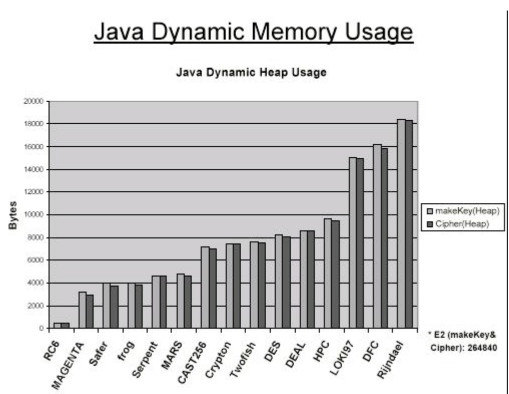
Class file size in Java



95

Avi Rubin - CS 600.443

Heap usage in Java



96

Avi Rubin - CS 600.443

More info on AES

- AES home page
  - http://www.nist.gov/aes
    - many papers
    - explanation of benchmarks
    - discussion of each cipher
    - history of selection process
    - conference information

Cryptography take aways

- A few problems are believed to be intractable:
  - Factoring is hard
  - Inverting RSA is hard
  - Computing discrete logarithms is hard
  - ….
- These beliefs are based on years of study and attempts to find fast solutions
- A crypto primitive or protocol should come with a proof that says *this primitive or protocol is as hard to break as it is to factor or invert RSA,…*
- The proof means that the security of the primitive or protocol is reduced to the security of a well-known, highly studied problem
- A primitive or protocol without such a proof has no guarantees—there is very little accumulated evidence that it is secure

Other Crypto Issues

- Key Agreement Protocols
- Pseudo random generators, Pseudo random functions
- One-way functions, Trapdoor functions
- Secure multiparty computation
- Zero-Knowledge proofs
- Models of adversaries/Definitions of security
- ……

Key distribution

- Needham and Schroeder protocol
  - Symmetric key only
  - Assumptions:
    - "Trusted" third party, T
    - Two principles, A and B
    - long-term shared keys between principles and TTP (K_{AT}, K_{BT})
    - Nobody reveals his/her secret key
  - Goals:
    - A and B wish to have K_{AB}
    - Nobody else should know K_{AB}
    - A and B believe that only they share the secret key

Needham and Schroeder

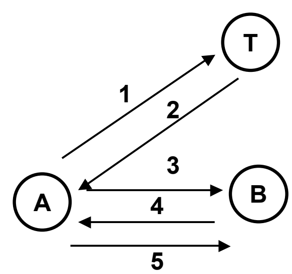
1. A -> T: A, B

2. T -> A: $\{B, K_{AB}, \{ A, K_{AB} \}_{K_{BT}} \}_{K_{AT}}$

3. A -> B: $\{ A, K_{AB} \}_{K_{BT}}$

4. B -> A: $\{N\}_{K_{AB}}$

5. A -> B: $\{N+1\}_{K_{AB}}$



Problems

- NS is intended to distribute short-term keys
- However, if a key is broken
  - can replay message 3 and force B to use old, compromised key
- B never proves knowledge of the key.
- How would you fix both these problems?

Kerberos

- Based on Needham and Schroeder
- Users authenticate by being able to decrypt ticket for tgt
- Early version susceptible to offline dictionary attack
- Tickets are issued for services
- Tickets expire
- Widely deployed and used system
- DCE is based on Kerberos

DH setting

- Alice
- Bob
- No previous contact between them
- Both have a computer
- Eve hears every single message between them

Can Alice and Bob communicate information to produce a secret key that Eve doesn't know?

(take a vote in the class)

Diffie-Hellman Key Exchange

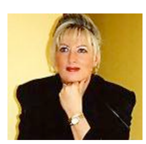
	Known to Alice	Public	Known to Bob
Choose secret values A, B	A	p, g	B
Compute and exchange public values g^A, g^B	$g^A \bmod p$ $g^B \bmod p$		$g^B \bmod p$ $g^A \bmod p$
Compute shared secret g^{AB}	$(g^B)^A \bmod p$ $= g^{AB} \bmod p$	$g^A \bmod p$ $g^B \bmod p$	$(g^A)^B \bmod p$ $= g^{AB} \bmod p$

105

Avi Rubin - CS 600.443

Play in the middle attack

- Alice and Bob pick random secrets a_i and b_i
- Problem is lack of authentication.
- Play-in-the-middle attack:


- Authentication and key exchange have to be tightly connected.
- One approach is to use public keys

106

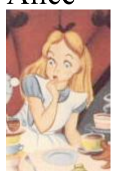
Avi Rubin - CS 600.443

Key Exchange: Establishing a (symmetric) Session Key

k

Bob

Alice



$pubkey_{Bob}$

$privkey_{Alice}$

$pubkey_{Alice}$

$privkey_{Bob}$

k

Avi Rubin - CS 600.443

107

Using cryptography

Important principles

- Don’t design your own crypto algorithm
  - Use standards whenever possible
- Make sure you understand parameter choices
- Make sure you understand algorithm interactions
  - E.g. the order of encryption and authentication
    - Turns out that authenticate then encrypt is risky
- Be open with your design
  - Solicit feedback
  - Use open algorithms and protocols
  - Open code? (jury is still out)

Building systems with cryptography

- Use quality libraries
  - SSLeay, lim (from Lenstra), Victor Shoup’s library, RSAREF, cryptolib
  - Find out what cryptographers think of a package before using it
- Code review like crazy
- Educate yourself on how to use library
  - Caveats by original designer and programmer

Common issues that lead to pitfalls

- Generating randomness
- Storage of secret keys
- Virtual memory (pages secrets onto disk)
- Protocol interactions
- Poor user interface
- Poor choice of key length, prime length, using parameters from one algorithm in another

Example: Web cookies

- Cookies were designed to offload server state to browsers
- Someone made a design choice
  - Use cookies to authenticate and authorize users
  - E.g. Amazon.com shopping cart, WSJ.com
- New design choice means
  - Cookies must be protected
    - Against forgery (integrity)
    - Against disclosure (confidentiality)
- Cookies not robust against web designer mistakes
  - Were never intended to be

Many security problems arise out of a technology built for one thing applied to something else incorrectly.

Sensus

- Created by Lorrie Craner and based on Fujioka, Okamoto, Ohta (FOO)
- Participants
  - Voter
  - Voter agent (totally trusted component, runs locally)
  - validator - ensure one vote per person
  - tallier - count ballots and report results

Not designed for Internet voting in public elections.

Blind signatures

- Need validator to sign m
- Validator should not know value of m
- Voter must be able to verify blind signature
- Assume RSA scheme
  - $n = pq$, where p and q are large primes
- Analogy of envelope with carbon paper in it
- Public exponent of validator is e, signing exponent is d

Blind Signatures (Chaum)

- All arithmetic is mod n
- Blinding (performed by voter):
  - choose a random blinding factor r
  - compute and present for signing: $m \times r^e$ where m is the message
- Signing (performed by validator):
  - compute $(m \times r^e)^d$
  - this is equal to $r \times m^d$
- Unblinding (performed by voter):
  - compute $r \times m^d / r = m^d$

Validator

- Public key pair: *ve*, *vd*
- Registered Voter List (RVL)
  - voter IDs
  - voter public keys
  - whether voter ballot has been validated (dynamically updated)

se, sd	ballot seal key
ie, id	voter key pair
ve, vd	validator key pair
te, td	tallier key pair
K	blinding factor

Tallier

- public key pair: *te*, *td*
- ve (validators public key)
- T: election tally
- Receipt List (RL)
  - list of receipts sent out
  - corresponding sealed ballots
  - decyption keys
  - recipt numbers

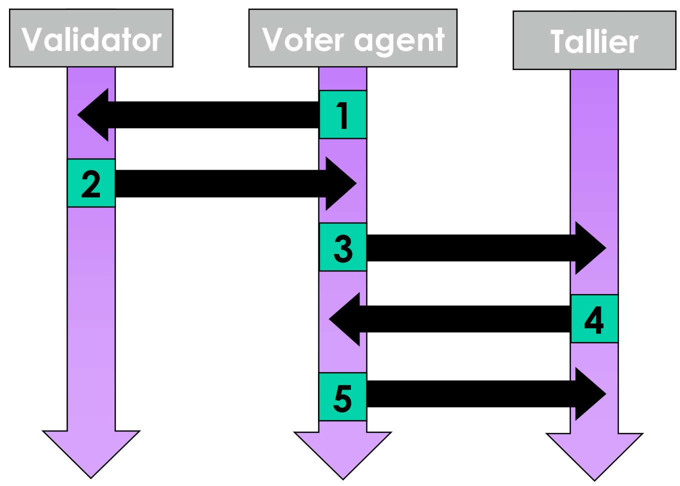
se, sd	ballot seal key
ie, id	voter key pair
ve, vd	validator key pair
te, td	tallier key pair
K	blinding factor

Voter agent

- V: a voted ballot
- ID, voter ID number
- ballot seal key pair: *se*, *sd*
- $m = V^{se}$ (sealed vote)
- k = large random number for blinding
- voter key pair: *ie*, *id*
- talliers and validator public keys: *te*, *ve*
- $b = m\ k^{ve}$ (blinded, sealed ballot)

se, sd	ballot seal key
ie, id	voter key pair
ve, vd	validator key pair
te, td	tallier key pair
K	blinding factor

The Sensus Polling Protocol



The Sensus protocol

- Voter agent sends validator, sealed with *ve*
  - *b*: blinded, sealed ballot
  - ID: ID number
  - b^{id}: *b*, signed with *id*, voter private key
- Validator unseals with *vd* and
  - verifies that $b = (b^{id})^{ie}$, checks the signature on *b*
  - updates RVL (registered voters list)
  - signs *b*
  - sends b^{vd} to voter agent, after sealing with *ie*
- Net result is that validator signs *b*

se, sd	ballot seal key
ie, id	voter key pair
ve, vd	validator key pair
te, td	tallier key pair
K	blinding factor

Sensus (cont.)

- Voter agent unseals with *id*
  unblinds b^{vd} by dividing by k
  obtains m^{vd} which is m, signed by validator
  verifies that $(m^{vd})^{ve} = m$
  sends (m^{vd}, V^{se}) to tallier, sealed with *te*
- tallier unseals with *td* and
  verifies: $V^{se} = (m^{vd})^{ve}$
  signs V^{se} to produce $(V^{se})^{td}$ = receipt
  updates RL with a receipt # and sealed ballot
  sends $(V^{se})^{td}$, receipt # to voter agent

se, sd	ballot seal key
ie, id	voter key pair
ve, vd	validator key pair
te, td	taller key pair
K	blinding factor

Sensus (cont.)

- Voter agent verifies receipt
  checks that $V^{se} = ((V^{se})^{td})^{te}$
  sends ballot secret key, *sd*, and receipt number to tallier
- tallier
  opens V^{se} with *sd*
  updates RL and T

se, sd	ballot seal key
ie, id	voter key pair
ve, vd	validator key pair
te, td	taller key pair
K	blinding factor

Evaluation of Sensus

- Accuracy
  - altered, eliminated, and invalid votes can be detected and corrected
- Democracy
  - if voters abstain, validator may submit ballots for them
  - these invalid ballots may be detected, but not corrected
- Privacy
  - not possible to link a ballot to the voter who cast it
  - does not prevent a voter from proving how he or she voted
    - Could potentially solve with last-vote-counts process
- Verifiability
  - voters can verify that their ballots were counted correctly and protest anonymously

Why inadequate for Internet voting

- Assumes communication occurs over an anonymous channel
- Machines (along with secrets on them) are secure
  - No Trojans, viruses, worms
  - Trusted O/S, applications, bug-free platform
- Assumes no subliminal channels in RSA
  - Depends on the implementation (no random padding)
- Assumes network is highly available
- Assumes there is a national registry of identities and public keys.
  - Assumption: election PKI is available in all places where it is adopted

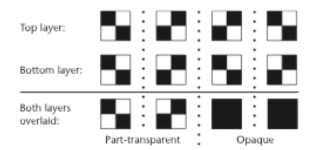
Chaum’s receipt system

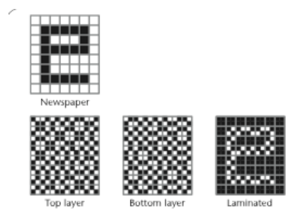
Properties

- Uses visual cryptography
- Generates receipts that only voter verifies
- Can check that receipt exists in final tally
- Cannot show anyone how you voted
- Requires some widely available public keys
- Requires trustees who perform mixing function

Encoding the ballot

- Two laminated sheets held together





127

Avi Rubin - CS 600.443

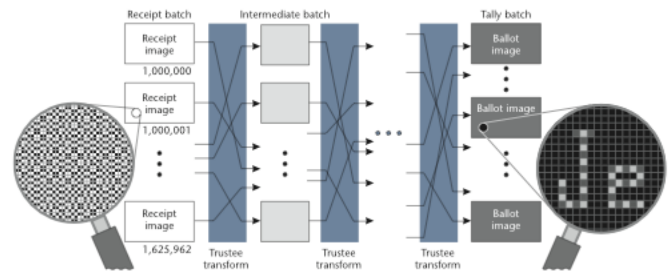
Ballots

- How ballots are generated
  - pick one layer at random
  - encode choices in other ballot
  - swap random number of equivalent pixel groups
    - (i.e.rotate grey boxes to change pixel locations)
  - which ones were swapped is the “key” kept by trustees
  - keep one layer
  - serial number printed on ballot to help voter lookup in tally
- Ways to cheat:
  - print an incorrect layer
  - use same serial number for 2 different receipts
  - skip a ballot in the tally
  - But, odds of getting caught are 50% per ballot
  - If enough voters check, say n, 1 in 2^n chance of getting away with it.

128

Avi Rubin - CS 600.443

Tallying



129

Avi Rubin - CS 600.443

Tallying (cont.)

- Uses analogy of Russian dolls
  - xor tricks used to break up ballot into layers, each encrypted with a trustee's private key
  - Each trustee only sees *dolls* at its layer
  - After *m* trustees decrypt their layer, the actual ballot is revealed.
  - All ballots can be printed on a web site along with all receipts
    - but not corresponding to each other
  - Voters can check that their ballot counted

130

Avi Rubin - CS 600.443