Chaffing & Winnowing
Confidentiality without Encryption!

Ronald Rivest
Cryptobytes, Summer 1998, p12-17

Introduction

- Chaff -> worthless parts
- To Winnow -> to separate out the chaff

- No Encryption, no decryption!
- No Export control?

Network Security (N. Dulay & M. Huth)

Public Key Cryptography & Digital Signatures (4.2)
Chaffing

- **Chaffing** -> adding fake packets with bogus MACs.
  - MAC based on sequence number and message.
- **Winnowing** -> discarding packets with bogus MACs

**Wheat** - Good packets
**Chaff** - Bad packets

1. Hi Bob, 462312
2. Meet me at, 782290
3. 7 PM, 238291
4. Love Alice, 839128

1. Hi Bob, 462312
1. Hi Larry, 388231
2. I’ll call you at, 562381
2. Meet me at, 782290
3. 7 PM, 238291
3. 6 PM, 823911
4. Yours Sue, 728377
4. Love Alice, 839128
Security

- Security depends on difficulty (for the adversary) of distinguishing the chaff from the wheat.

- Chaffing will normally add at least one chaff packet for each wheat packet.

- We also need to make wheat packets unintelligible. How can we do this?
Make Wheat packets a single byte or bit!

- Which bits are bogus?
- Which bits are authentic?
- Note: adding a bogus MAC is trivial, just add a random number (e.g. 64 bits).
- For efficiency would like to process many bits per packet instead of one or 8.
- Can use a so-called package transform that transforms message such that receiver can only produce original if he receives entire transformed message (transform is keyless).
Variations on a Theme

- Creating chaff can be done by a 3rd party, who doesn’t know any secret MAC keys!
- Alice & Bob may not even be aware of the chaff maker!
- Charles the Chaff Maker could multiplex Alice-to-Bob packets and David-to-Elaine packets.
- Alice can hide messages. So could use 2 (or more MACs). If asked to reveal MAC, she reveals MAC for innocuous message.
Public Key Cryptography & Digital Signatures

Michael Huth
M.Huth@doc.ic.ac.uk

www.doc.ic.ac.uk/~mrh/430/
Introduction

- Also known as ASYMMETRIC KEY, TWO KEY encryption
- Based on mathematics rather than on substitution or transposition ciphers
- Different keys for encryption and for decryption.
- One key is kept private (secret), the other can be made public (not secret). Private and Public keys are related mathematically

PUBLIC-KEY vs SYMMETRIC-KEY

- Public-Key cryptography is not “better” than Symmetric-Key
- Symmetric-Key cryptography is much faster than Public-Key version -> up to 1000 times faster
- Public-Key cryptography used for: authentication, digital signatures, and key-management
- Symmetric-Key cryptography used for: bulk encryption, e.g. high-volume packet flow on network
Merkle's Puzzles (has applications in electronic contract signing)

1 Million Encrypted Puzzles. Each encrypted with a diff. 20-bit key, each holds a diff. 128-bit key. Puzzle order is random.

PuzzleNo: 6248
128-bit Key: 10010...01101

E_{K_{20}}(6248, K_{128})

Randomly picks 1 puzzle to crack, cracks it

PuzzleNo: 6248
Msg: E_{K_{128}}("Hello")

6248, E_{K_{128}}("Hello")

Looks up 128-bit key for Puzzle 6248

Attacker needs to crack ~ 0.5 million puzzles!
Confidentiality (Secrecy)

Receiver’s (Bob’s) public key

Receiver’s (Bob’s) private key

Network Security (N. Dulay & M. Huth)

Public Key Cryptography & Digital Signatures (4.10)
Authentication

Sender’s (Alice’s) private key

Sender’s (Alice’s) public key

Network Security (N. Dulay & M. Huth)

Public Key Cryptography & Digital Signatures (4.11)
Confidentiality with Authentication

Sender’s (Alice’s) private key  

sign

Receiver’s (Bob’s) public key

C

Receiver’s (Bob’s) private key  

verify

Sender’s (Alice’s) public key

P

Public Key Cryptography & Digital Signatures (4.12)
Requirements for Public-Key Cryptography

- Should be easy & efficient to generate keys (public and private)
- Should be easy & “efficient” to encrypt and decrypt
- Should be hard to compute PRIVATE-KEY from PUBLIC-KEY
- Should be hard to compute PLAINTEXT from PUBLIC-KEY and CIPHERTEXT
- Encryptions, Decryption might be applied in either order

ONE-WAY FUNCTION $f$

- $C = f(P)$ “Easy”
- $P = f^{-1}(C)$ Infeasible

TRAP-DOOR 1-WAY FUNCTION $f$

- $C = f(K, P)$ “Easy” if K & P known
- $P = f^{-1}(K, C)$ “Easy” if K & C known
- $P = f^{-1}(K, C)$ “Infeasible” if K not known, C known

Network Security (N. Dulay & M. Huth)
Diffie-Hellman Key Exchange

- A and B agree on two numbers \((r, p)\)
- \(r\) and \(p\) can be public

- A picks a secret number \(a\)
- B picks a secret number \(b\)

- A computes \(f(r, p, a)\) Call result \(f_A\)
- B computes \(f(r, p, b)\) Call result \(f_B\)

- A sends B the value \(f_A\)
- B sends A the value \(f_B\)

- A computes \(f(f_B, p, a)\) call result \(k_A\)
- B computes \(f(f_A, p, b)\) call result \(k_B\)

Correct protocol should ensure that \(k_A = k_B\)

\(k_A\) & \(k_B\) should be hard to predict, knowing \(r, p, f_A, f_B\) and \(f\)

But what is \(f\) ?
Exercise

- Try out $k_A$ and $k_B$ for $f(r, p, x) = r + p + x$, $r = 3$ and $p = 12$

- Now crack $f(r, p, x) = (r \times x) \mod p$, $r = 4$, $p = 10$, $f_A = 8$, $f_B = 6$
Discrete Logs

- Find $x$ where $3^x = 13 \mod 17$
- Find $x$ where $3^x = 7 \mod 13$

For:

$$g^x = f \mod p$$

- given integers $g$, $f$ and prime $p$ it is computationally expensive to calculate the discrete log $x$ if $p$ is large, e.g. $> 200$ digits

... (and if $p$ meets some other conditions hold that shield against known discrete-logarithm attacks)
Background: primitive roots

- Let $n$ be a natural number.
- Recall: $m$ is co-prime to $n$ iff greatest common divisor of $n$ and $m$ is $1$.
- Example: $5$ and $6$ are co-prime whereas $3$ and $6$ are not.

- Definition: A primitive root modulo $n$ is some integer $g$ such that any number co-prime to $n$ is some power of $g$ modulo $n$.
- Formally, $g$ is primitive root of $n$ iff for all $m$ with $\gcd(n,m) = 1$, there is some $k$ such that $g^k = m \pmod{n}$.

- FACT: Every prime number has a primitive root.

- Example: $3$ is a primitive root of $7$. (Verify this.)
**Diffie-Hellman Key Exchange**

\[
f (r, p, x) = r^x \mod p
\]

- \( r = 5 \), \( p = 563 \)
- \( r \) is a primitive root of \( p \)
- \( p \) is a very large prime

\[
f (r, p, x) = r^x \mod p
\]

- \( a = 9 \)
- \( f_A = r^a \mod p \)
  \( f_A = 5^9 = 78 \mod 563 \)

- \( b = 14 \)
- \( f_B = r^b \mod p \)
  \( f_B = 5^{14} = 534 \mod 563 \)

- \( k_A = 534^9 = 117 \mod 563 \)
- \( k_B = 78^{14} = 117 \mod 563 \)

- \( k_A = (f_B)^a \mod p \)
  \( k_A = (r^b)^a \mod p \)
  \( k_A = r^{ab} \mod p \)

- \( k_B = (f_A)^b \mod p \)
  \( k_B = (r^a)^b \mod p \)
  \( k_B = r^{ab} \mod p \)
Question

To ensure the correctness of the Diffie-Hellman exchange protocol:

What is the key algebraic property of the function that sends $x$ to $r^x \mod p$?
RSA (Rivest-Shamir-Adleman)

- Observation: easy to multiply two large primes, p and q:
  \[ p \times q = N \]

- Very difficult to factorise N back into p and q.

- Instrumentation of plain-text prior to encryption:
  Split large messages into blocks less than \( p \times q \) in value, e.g., for \( N = 7 \times 5 \) you could use 4-bit blocks

- RSA patent expired in September 2000.

- Hardware RSA 1000 times slower than hardware DES.
  Software RSA 100 times slower than software DES
  (similar factors for AES)

- Remaining question: how to efficiently generate large prime numbers p and q?
  (You are welcome to research this. E.g., Miller-Rabin randomized algorithm.)
RSA

KEY GENERATION

- Pick two large primes, p and q (keep p and q secret)
- Calculate \( N = p \times q \) (result not secret)
  - Calculate \( \phi(N) = (p-1) \times (q-1) \)
- Calculate E and D such that \( E \times D = 1 \mod \phi(N) \).
  - E and D must be co-prime to \( \phi(N) \)
- Public Key = \((E, N)\)
  - Private Key = \((D, N)\)

ENCRYPTION

- \( C = P^E \mod N \)

DECRIPTION

- \( P = C^D \mod N \)

Example:

KEY GENERATION

- \( p = 47, \quad q = 71 \)
  - \( N = 47 \times 71 = 3337 \)
  - \( \phi(N) = (47-1) \times (71-1) = 3220 \)
- \( E \times D = 1 \mod 3220 \)
  - e.g. Encryption key \( E = 79 \)
  - e.g. Decryption key \( D = 1019 \)
- Public Key = \((79, 3337)\)
  - Private Key = \((1019, 3337)\)

ENCRYPT \( P = 688 \)

- \( C = 688^{79} \mod 3337 = 1570 \)

DECRYPT \( C = 1570 \)

- \( P = 1570^{1019} \mod 3337 = 688 \)
RSA

- p and q can be destroyed after N, E, and D are generated. Useful for private key holder to keep p and q for faster decryption - can use the Chinese Remainder Theorem to perform calculations mod p and mod q instead of mod pq.

- $\text{Ø}(N)$ is Euler's totient function applied to $N$: the number of numbers less than N that are co-prime to N, e.g. $\text{Ø}(p\times q) = (p-1)(q-1)$.

- Relatively prime means “co-prime”, i.e. that $x$ and $y$ share no common factors, i.e. $\text{GCD}(x,y)=1$.

- If $\text{Ø}(N)$ can be computed efficiently from $N$, this breaks RSA, how so?
**Explanation of equations relating D and E**

1. **Equations**:
   - \( C = P^E \mod N \)
   - \( P = C^D \mod N \)
   - \( P = (P^E)^D \mod N \)
   - \( P = P^{ED} \mod N \) (\( P < N \))

2. **Can we find E, D and N such that**
   - \( P = P^{ED} \mod N \)

   and also such that it is “Easy” to compute \( P^E \) and \( C^D \), but infeasible to get D given E and N?

3. **Euler showed that**
   - \( P = P^{k\cdot\phi(pq)+1} \mod (pq) \)

4. **So**
   - \( ED = k\cdot\phi(pq) + 1 \)

5. **Which is equivalent to**:
   - \( ED = 1 \mod \phi(pq) \)
   - \( D = E^{-1} \mod \phi(pq) \)
   - \( D = E^{-1} \mod (p-1)(q-1) \)
Issues

IMPLEMENTATION

- How to pick two large random primes? (Algorithms are based on probabilistic testing).
- How to derive E & D?
- How to quickly compute $X^Y \mod N$? (Iterative squaring)

UNCONDITIONAL SECURITY

- A public-key cryptosystem can never provide Unconditional Security! Why?

ATTACKS

- Factorise N. Very hard if N is large, e.g. if N has 1024 bits we would need $10^{15}$ MIP years. For N having 431 bits, about 500 MIP years.
- Reverse engineer and try to break prime-number generator.
- Timing attacks (ciphertext only). Determine how long decryption operations take and make inferences on resulting bits. Counter-measures? Code obfuscation, etc.
Use RSA to pass an encrypted AES session key, then use AES to encrypt-decrypt subsequent messages.
Authentication, Integrity, Non-repudiation

**AUTHENTICATION**
- Verify the source of a message.
  -> MACs, Digital Signatures

**INTEGRITY**
- Has message been tampered with?
  -> MACs, 1-Way Hash Functions, Digital Signatures

**NON-REPUDIATION**
- Prevent sender/receiver from later denying sending/receiving a message.
  -> Arbitrated Digital Signatures, Digital Certified Email, Simultaneous Contract Signing

- **Message Authentication Code (MAC):** think of it as a key-based hash function
- **One-Way Hash Functions**
  e.g. MD5, SHA (Secure Hash Algorithm)
- **Digital Signatures**
  e.g. RSA, DSS (Digital Signature Standard)
- **Authentication Protocols**
Symmetric-Key Authentication

AUTHENTICATION

- Only sender and receiver know secret key. If decrypted message appears "valid" then sender must have sent it?
- What if message is binary data or is hard to "validate"?
- We could add a checksum and/or other checks to message (e.g. sequence numbers, timestamps) to reduce chances that we do not accept a bogus message
- Such authentication protects A and B from C, but not from each other -> Arbitration, Authentication Protocols.

Message Authentication Code (MAC)

- Also known as a Cryptographic Checksum. Provides Authentication and Integrity
- Use a secret key to generate a small output block from Message e.g. use AES in CBC mode; last encrypted block acts as a MAC.
- Best if MAC key is different from secret key
- MACs are similar to, but not identical with, hash functions

Network Security (N. Dulay & M. Huth)
Digital Signatures

- **AUTHENTIC**: Can verify who signed msg
- **UNFORGEABLE**: Only signer could have signed msg
- **NOT REUSABLE**: Cannot bind signature to different msg
- **UNALTERABLE**: Cannot alter msg without affecting its signature
- **CANNOT BE REPUDIATED**: Signer cannot later deny signing.

Hand-written Signatures:

Do they satisfy these properties?

Are their features needed for digital signatures that have no equivalent or need for hand-written signatures?
Digital Signatures

- Should be message-dependent
- Should be sender-dependent
  (Dependent on information unique to sender)
- Should be easy to produce
- Should be easy to verify
- Should be infeasible to forge
- Useful if signatures can be stored separately from signed message

- Digital Signatures can be used for encrypted and unencrypted messages.
- Symmetric-Key digital signature schemes exist but are not elegant
  (We won’t cover them.)
- Encrypting a message with one’s RSA private key yields a digital signature,
  but signature is as long as the message!
One-Way Hash Functions

- Also known as SECURE HASH FUNCTIONS, MESSAGE DIGESTS
- Provide INTEGRITY only, no key needed (unlike MACs)

**EXAMPLES**
- MD5 (Message Digest 5)
- SHA (Secure Hash Algorithm)

### One-Way Hash Function

- Plaintext P (e.g. 10Mb) → One-Way Hash Function → Hash Value H (e.g. 256 bits)

- Easy to produce H from P
- Size of P >> Size of H (Compression)
- Infeasible to produce P from H
  - find P2 such that Hash(P2) = H (Weak Collision Resistance)
  - find P1 & P2 such that Hash(P1) = Hash(P2) (Strong Collision Resistance)
  - Infeasible to find P2 such that Hash(P2) = H (One-Way)
Birthday Attack

- If hash value is too short then we can employ a birthday attack.

- If an element can take on \( N \) different values, then we can expect a collision after about \( \sqrt{N} \) random elements. For \( N = 2^M \), we have a collision after about \( 2^{M/2} \).

- Given a hash value \( H \), to find \( P \) such that \( \text{Hash}(P) = H \) requires \( 2^M \) random messages.

- However to find two \( P_1, P_2 \) such that \( \text{Hash}(P_1) \) and \( \text{Hash}(P_2) \) produce the same hash value only requires \( 2^{M/2} \) random messages.

- Generate \( 2^{M/2} \) variations of non-fraudulent message

- Generate \( 2^{M/2} \) variations of fraudulent message

- Probability of finding a non-fraudulent/fraudulent pair > 0.5

- Note we can insert space-space-backspace-newline (etc.) sequences to generate variations

- CONCLUSION: Use lengthy hash values, e.g. 256 bits

- Caution: even 265 bits may be insecure, e.g. attacks on SHA-1
Digital Signature

- Common technique is to encrypt a one-way Hash Value with Signer’s Private (Signing) Key -> **AUTHENTICATION + INTEGRITY**

  - Timestamps are often hashed along with message P. Why?

  **Plaintext** $P$ → **1-Way Hash Function** → **Encrypt** → **Signature** $S$
Verifying a Digital Signature

<table>
<thead>
<tr>
<th>Alice's Public Key</th>
<th>Decrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature $S$</td>
<td></td>
</tr>
<tr>
<td>Plaintext $P$</td>
<td>1-Way Hash Function</td>
</tr>
</tbody>
</table>

Hash Value H1

$H1=H2?$

Hash Value H2
Signed Messages

**SIGN**

\[\text{AlicePrivK} \rightarrow \text{HF} \rightarrow \text{E} \rightarrow \text{Append} \rightarrow \text{P, S} \]

**VERIFY**

\[\text{AlicePubK} \rightarrow \text{Split} \rightarrow \text{D} \rightarrow \text{P} \rightarrow \text{P, S} \rightarrow \text{HF} \]

Network Security (N. Dulay & M. Huth)
Sending a Signed Encrypted Message

Bob’s publicK

\[ \text{sessionK} \rightarrow \text{RSA} \rightarrow C1 \]

Bob’s privateK

\[ C1 \rightarrow \text{RSA} \rightarrow \text{sessionK} \]

\[ P, S \rightarrow \text{AES} \rightarrow C2 \]

\[ C2 \rightarrow \text{AES} \rightarrow P, S \]

\[ \text{AlicePrivK} \]

\[ \text{Sign} \rightarrow P \]

\[ \text{AlicePubK} \]

\[ \text{Verify} \rightarrow P \]

Network Security (N. Dulay & M. Huth)
Public-Key Certificates

- Combination of a name and Public Key signed by a more trusted, third party

- Certificates include additional information, e.g. expiration date of public key. What other information would be of interest?

Network Security (N. Dulay & M. Huth)
Disavowed Signatures

- **What if the sender wants to disavow sending a message**, e.g. the sender anonymously publishes her private key and then falsely claims that her private key was lost or stolen and that someone else has forged her signature on sent messages e.g. a contract to buy shares which then fall in price?

- **Timestamp messages?**

- **Best if Private Keys are held in tamper-proof devices.**
Arbitrated Digital Signature (Unencrypted)

1) A → T: idA, signA (idA, signA (P))
   T: check (idA)
   verifyA (idA, signA (P))

2) T → B: signT (timestamp, idA, signA (P))
   B: verifyT (timestamp, idA, signA (P))
   check (idA)
   check (timestamp)
   verifyA (P)

- T should also send the second message to A. If A did not originate it, A should own up immediately.
Some variants of digital signatures

- **Blind Signatures**
  Sign documents without seeing contents.

- **Undeniable Signatures**
  Cannot be verified without signer's consent.

- **Proxy Signatures**
  Pass power to sign to someone else.

- **Group Signatures**
  Any member of a group can sign.

- **Fail-stop Signatures**
  Each public key has many private keys.

- **Simultaneous Contract Signing**
  Neither party wishes to sign unless others sign as well i.e. simultaneously

- **Digital Certified Mail**
  Receiver must sign a receipt before reading mail

- **Secure Electronic Voting**

- **Digital Cash**

- **Zero Knowledge, Anonymous Broadcast, Encrypted Computation**
Elliptical Curve Cryptography

Michael Huth
M.Huth@doc.ic.ac.uk

www.doc.ic.ac.uk/~mrh/430/
Reading

- Stallings Chapters 9/10/11
Elliptical Curve Cryptography (ECC)

- A possible successor to RSA (and DSA).
- Already being adopted in various standards.
- For a (believed to be) similar level of security, ECC public/private keys are much smaller than RSA keys

<table>
<thead>
<tr>
<th>Time to Break</th>
<th>RSA keys</th>
<th>ECC keys</th>
<th>RSA/ECC Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS years</td>
<td>keysize</td>
<td>keysize</td>
<td></td>
</tr>
<tr>
<td>$10^4$</td>
<td>512</td>
<td>106</td>
<td>5:1</td>
</tr>
<tr>
<td>$10^8$</td>
<td>768</td>
<td>132</td>
<td>6:1</td>
</tr>
<tr>
<td>$10^{11}$</td>
<td>1024</td>
<td>160</td>
<td>7:1</td>
</tr>
</tbody>
</table>
## ECC Speed

<table>
<thead>
<tr>
<th>Function</th>
<th>ECC-163 (msecs)</th>
<th>RSA-1024 (msecs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign</td>
<td>2.1</td>
<td>228</td>
</tr>
<tr>
<td>Verify</td>
<td>9.9</td>
<td>12.7</td>
</tr>
<tr>
<td>Key Generation</td>
<td>3.8</td>
<td>4708</td>
</tr>
<tr>
<td>Key Exchange</td>
<td>7.3</td>
<td>1654</td>
</tr>
</tbody>
</table>
Basic Idea

- Take elliptic Curves of the form:
  \[ y^2 = x^3 + ax + b \mod p \]
  (a, b are constants, p a large prime)

- If P=(x,y) satisfies the equation, then P is a "point on the curve"
- Points can be "added" to generate a new Point. Note: Point addition is not simple number addition but a "non-linear" operation.
  \[ R = P + Q \]

For \[ Q = nP \]
(nP is P added to itself n times)

- It is relatively easy to calculate \( Q \) given \( n \) and \( P \)
- But is very hard to calculate \( n \) given \( P \) and \( Q \).
- For more details see Stallings.
Interlude: Modular Arithmetic

Congruence \( a \equiv b \mod n \)

means \( a/n \) and \( b/n \) have the same remainder

- \((a+b) = (b+a) \mod n\), \((a*b) \mod n = (b*a) \mod n\)
- \([(a+b)+c] \mod n = [(a+(b+c)) \mod n\]
- \([a*(b+c)] \mod n = [(a*b)+(a*c)] \mod n\)
- \(a + b \mod n = [(a \mod n) + (b \mod n)] \mod n\)
- \(a - b \mod n = [(a \mod n) - (b \mod n)] \mod n\)
- \(a * b \mod n = [(a \mod n) * (b \mod n)] \mod n\)
- \(a^b \mod n = [a \mod n]^{b-1} \mod n\)
- \(a^{bc} \mod n = [a^b \mod n]^c \mod n\)

if \( A \times B = 1 \mod n \)
then \( A \) and \( B \) are multiplicative (or modular) inverses \( \mod n \). Often written:

\[
A^{-1} = B \mod n \quad \text{or} \quad B^{-1} = A \mod n
\]
Euclid’s Algorithms

```python
def gcd(a, b):
    # find greatest common divisor of a, b
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

def ExtendedGcd(a, b):
    # returns gcd(a,b), inv of a, inv of b
    if b == 0:
        return a, 1, 0
    else:
        x, y, z = ExtendedGcd(b, a % b)
        return x, z, y - int(a/b)*z

gcd550_1759, inv550, inv1759 = ExtendedGcd(550, 1759)
print "inverse of 550 mod 1759 is ", inv550 % 1759
```