# Course Outline

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem description/challenges</td>
<td>cryptographic protocols</td>
<td>general principles PHY-layer</td>
<td>sensor networks (IEEE 802.15.4)</td>
<td>wireless networks</td>
</tr>
<tr>
<td>introduction to cryptography</td>
<td>details of some deployed cryptosystems</td>
<td>details of some deployed protocols (IEEE 802.11)</td>
<td>big exercise (lab)</td>
<td>paper presentation</td>
</tr>
</tbody>
</table>
Acknowledgment

- Course material for Monday and Tuesday (Cryptography) is based on the Security Principles Course, Software and System Security, Software Engineering Programme (SEP). Prof Andrew Martin.

- Course material for Wednesday (Wireless Security) – Friday is based on Security in Wireless Network Course, Software and System Security, Software Engineering Programme (SEP). Prof Ivan Martinovic
Technical Element

- cryptography and protocols
  - fairly abstract
  - mathematically deep: we will only scratch the surface
  - important controls, but not necessarily central to every security regime
  - underpin a great deal of technologies
  - indicative of the level of complexity in typical technical controls
  - ‘typical’ indication of skills and approaches needed by attackers and defenders
  - more long-lived insight than we would gain from studying particular products or systems
Acknowledgment

- Course material for Monday and Tuesday (Cryptography) is based on the Security Principles Course, Software and System Security, Software Engineering Programme (SEP). Prof Andrew Martin.

Contents

- Cryptography
  - history and concepts
- Substitution vs Transposition
- Block ciphers vs Stream ciphers
- One-way functions
- Symmetric encryption and block modes
- Asymmetric Encryption

Cryptography's long history

- For centuries, cryptography has been employed for state secrets
- Apparently, Julius Caesar used a cipher:

<table>
<thead>
<tr>
<th>Key=D</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>...</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>...</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

  - thus:

  | F | U | B | S | W | R | J | U | D | S | K | B |

- ‘limited’ usefulness: why?
**Vigenère (1586)**

- generalizes Caesar's cipher by using a different key letter for each encryption
- Many more possible keys
- Frequency analysis harder
- Basis of Enigma machine (key lengths in excess of $26^3$)
- Related to an embarrassingly-weak cipher that's still used sometimes
- How does it improve on Caesar's cipher?

**Transposition**

- keep the same letters, but re-order them

**Terminology**

- For centuries, cryptography has been employed for state secrets
- Apparently, Julius Caesar used a cipher:

<table>
<thead>
<tr>
<th>Key = D</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>...</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>...</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

- 'limited' usefulness: why?

**More terminology**

- replace letters, digits, blocks, by other letters, digits, blocks in a systematic but secret way; also known as cryptographic algorithm

- just about all ciphers depend on a secret key: the size (bit length) of the key is of interest, as is the key space: the set from which all possible keys are drawn
Attacks

- objective is (generally) to discover the key

Modern cryptography

- (most) modern cryptography relies on the same principles as the techniques known in antiquity
- replace letters with blocks of binary data
- use a mathematical function (or, equivalently, a circuit) in place of a lookup table: why?

Security versus Obscurity

- All respectable cryptography assumes that the attacker knows the algorithm (the cipher, and the implementation details) used for encryption (but not the key).
- Why is this?

Randomness

- The ideal cipher is indistinguishable from a random function
  - every ciphertext is equally likely
- this is known as the ‘random oracle’ model of cryptography
- true randomness is elusive: pseudorandom describes a function which passes suitable statistical tests
- we seek key-based mathematical functions which transform inputs in a pseudorandom way, when supplied with a randomly-chosen key
- if your attackers have the same pseudorandom number generator as you, they will use it to guess your keys

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.  John von Neumann
**Objective**

- the role of the randomness is to make ‘brute force’ attacks difficult
- given the way that each generation of technology gives rise to faster computers (c.f. Moore’s Law) and the increasing use of parallelism, we must build in large margins

**Entropy**

- Entropy is the measure of information content in a message.
  - It is measured in bits.
  - The entropy of message $M$ is sometimes written $H(M)$
  - (don’t confuse this with a hash! – see later)

Examples:
- a 1-bit field holding 0 or 1 for false/true has entropy 1 bit
- a 3-bit field reporting the day of the week has entropy somewhat less than 3 bits (more precisely, $\log_2 7$ bits)
- string field holding the values "true" and "false" also has entropy 1 bit
- in order to achieve entropy of 64 bits for a password field, we need 11 randomly chosen case-sensitive alphanumeric characters

**Question**

- Why does entropy (measuring information content) matter in our discussion of randomness?

**Recommended Paper:**
Communication Theory of Secrecy Systems By C. E. Shannon

**Substitution Ciphers**

- ’like’ Caesar’s cipher, but with a much bigger ‘alphabet’
  - for example, 26th possible ‘letters’
- simple substitution cipher is one-to-one plaintext to ciphertext
  - frequency analysis still applies
- homophonic (one to many), polygram (group to group), and polyalphabetic (multiple simple) ciphers all aim to confound frequency analysis
  - not necessarily successfully
03 Cryptography in Principle

Block Cipher

• Our general goal is to define $f$
  
• A lookup table would be ideal, but impractical (why?)
  
• So $f$ must be a mathematical function
  
• Bit sizes are examples
    - Plaintext size will generally match ciphertext size
    - Key size may differ, but will be similar order of magnitude to block size

Candidate: XOR

• Write $a \oplus b$
  
  $0 \oplus 0 = 0$
  $1 \oplus 1 = 0$
  $1 \oplus 0 = 1$
  $0 \oplus 1 = 1$
  
• Generalize for any number
    - Express in binary
    - Do bitwise $\oplus$
  
  $\forall a : 2 . \ a \oplus a = 0$
  $\forall a, b : 2 . \ a \oplus b \oplus b = a$

We could try $f = \oplus$

$\text{cipherText} = \text{plainText} \oplus \text{Key}$

$\text{plainText} = \text{cipherText} \oplus \text{Key}$

Block cipher for whole message

• Simplistic view: we will come back to this

Security Principle: A cipher should exhibit the avalanche effect: a change to a single bit of the input or the key should result in a potential change to every bit of the output.

• People do have a habit of screwing this up
  
• ‘Super-fast encryption’ often means ... XOR
  
• Windows CE (codename 'Pegasus')/ActiveSync 2.x offered to hold your NT password securely for you. It did this by storing
    - Password $\oplus$ susageP
Stream Cipher

- We are mainly considering *block ciphers*
- an entirely different construction is a *stream cipher*
  - encrypt a ‘continuous’ stream of data, rather than separating into blocks
  - uses XOR at its centre – so *keystream generator* must avoid repeating sequences
  - ideally suited to streaming media
  - different algebra; different concerns; same basic strength when done right

One-time pad: absolute secrecy

- classically: have a pad of randomly chosen letters; use each one once
- equivalent to Vigenère with an arbitrary long key, and a different key for each message
- provided the pad is truly random every ciphertext is equally likely, so without the pad it is impossible to recover the plaintext
- equivalent to a stream cipher with infinite key stream
- this is the only route to perfect secrecy
- truly random sources are hard to come by; and hard to share

Aside: transposition

- key-dependent *rearrangement* of bits has considerable strength
- limited random-access memory has curtailed its use
Notation

- Cryptography as function application:
  - $C = encrypt(P)$
  - $FUBSWRJUDSKB = caesar\left(CRYPTOGRAPHY\right)$
  - $C = encrypt(k, P)$
  - $FUBSWRJUDSKB = caesar\left(D, CRYPTOGRAPHY\right)$
  - $P = decrypt(k, C)$
  - $decrypt = encrypt^{-1}$

- Encrypted content in protocols etc.
  - 'message $m$ encrypted with key $k$'
  - ${m}_k$
  - Crypto algorithm determined from the context!

Forms of encryption

- One-way functions
  - passwords
  - hashes and message digests
  - hash-based authentication

- Symmetric encryption
  - Shared secret keys
  - Bulk message encryption

- Asymmetric encryption
  - Separate public-private key pairs
  - Key distribution
  - Message authentication and integrity

Password Security

- General principle: store encrypted passwords, only
- So the encryption does not need to be reversible
- Compare UNIX and Windows approaches: salt or no salt
- Passwords are now seen as very weak protection mechanisms
- Social issues abound: see PAS module

Question

Why do you 'need' a long and complex password for your email login, but only a 4-digit PIN for your bank ATM card?
Threats to passwords

- Direct guess of an individual’s password
- ‘Brute force’ attack
  - against an individual
  - against any/all the users of a system

- Modes of attack
  - ‘online’: guessing etc. against the live interface
    - best scenario for defender
  - ‘offline’: guessing undertaken against a copy, not the live system
    - circumvents many controls; avoids arousing suspicion

Password brute force

1. encrypt every possible password, and store the results in a look-up table (‘Rainbow table’)
2. obtain a copy of the password file for the victim system
3. for each encrypted password, use the look-up table to discover its plaintext version

- needs substantial storage, and one-off compute power
- returns passwords in negligible time
- works well against Windows XP (and all previous versions)
  - NT Lan Man had certain other related weaknesses, too

Salt vs No Salt

<table>
<thead>
<tr>
<th>username</th>
<th>encrypted password</th>
</tr>
</thead>
<tbody>
<tr>
<td>charlotte</td>
<td>e(charlotte’s password)</td>
</tr>
<tr>
<td>bob</td>
<td>e(bob’s password)</td>
</tr>
<tr>
<td>alice</td>
<td>e(alice’s password)</td>
</tr>
</tbody>
</table>

- $e(p)$: fixed encryption/hash function for passwords
- simple password file/database:
  - attacker can easily pre-compute encrypted version of all the passwords of a given length (say, $n$)
  - lookup table will be approximately $(n^80^n$ bytes
    - assuming there are 80 available characters to be typed in passwords
  - so for $n$ characters of strength, the user must remember $n$ characters

Salt vs No Salt

<table>
<thead>
<tr>
<th>username</th>
<th>salt</th>
<th>encrypted password</th>
</tr>
</thead>
<tbody>
<tr>
<td>charlotte</td>
<td>gv</td>
<td>e(gv</td>
</tr>
<tr>
<td>bob</td>
<td>A%</td>
<td>eA%(bob’s password)</td>
</tr>
<tr>
<td>alice</td>
<td>=k</td>
<td>e-k</td>
</tr>
</tbody>
</table>

- $e(p)$: fixed encryption/hash function for passwords
- salted password file/database:
  - salt value chosen at random when password is created: stored in cleartext
  - no significant overhead for legitimate normal use
    - nor for brute force approach (a)
  - lookup table has to be two characters longer: size $(n+2)80^n+2$ bytes
  - $n$ characters of ‘strength’ for $(n-2)$ characters of Alice’s memory
Passwords: summary

- offline brute-force guessing is the worst attack
  - and eventually always fatal
  - even an attacker with negligible resources can do this easily
- many legacy systems are subject to rainbow table attacks
  - so those systems are trivial to circumvent
- online attacks are harder to mount
  - but distinguishing good login attempts from bad ones is quite hard
  - as is knowing what to do about it
  - few systems implement a 'lock out' anyway

Message Digests

- hash: compact representation of large amount of data
- many-to-one: there are inevitable hash collisions

Hashing

- to construct a message digest, could we just XOR the blocks of the message together? Would a checksum suffice?
- hash: compact representation of large amount of data
- many-to-one: there are inevitable hash collisions
- cryptographic hash design goal
  - efficiency: make it easy to compute hash from message
  - one-way function: make it hard (i.e. effectively impossible) to compute message from hash
  - unpredictable collisions: make it hard (i.e. effectively impossible) to find two messages with the same hash
  - like a cipher: every input bit affects every output bit; whole output space should be reachable (and equally likely?)
**Message Authentication Code**

- Intending to give authentication without secrecy
- Avoid running costly encryption/decryption whenever possible
- Use a key-dependent one-way hash function
- Key not passed with the communication: recipient knows the key, and uses it to recompute the hash, and check its value.

**Symmetric Encryption**

- Historically, the only kind
- If you can do encryption, you can do decryption, and vice versa
- Same key is used for both
- Usually, run algorithm ‘in reverse’ for decryption
- Sometimes (DES) same algorithm used to encrypt and decrypt
  - Easy re-use in hardware
- Often called ‘secret key encryption’
  - Essential that key is kept secret

**Using block ciphers**

- Recall this picture
- The ‘obvious’ way to use this to encrypt a long message is to break it into blocks, and use \( f \) to encrypt each block separately
- This is called electronic codebook mode (ECB)
- Named because you could create a codebook (lookup) for blocks, but it would need \( 2^{128} \) entries (for a 128-bit block size)

**ECB visualized**

- Diagram showing the process of encryption using ECB mode.
Attacking ECB

- cryptanalyst who has plaintext and ciphertext for a few messages, can begin to compile a code book without knowing $k$ (or even $f$, actually)
- messages tend to have standard formats
- block replay problem

Cipher block chaining (CBC) mode: encryption

- requires initialization vector $iv$ (can be sent in cleartext at the start of the message)
- CBC is much 'safer'.
- problems with error propagation (not a problem with modern error-correcting channels)
  - bit errors, not too bad;
  - synchronization errors fatal
- still a basic correspondence between blocks
  - can you add/remove some at the end? depends on message structure
- very long messages still have patterns
- does not protect integrity very satisfactorily
- CBC is one of a number of safer ways to use a block cipher
Uses of symmetric cryptography

- secrecy: bulk encryption
- authenticity, integrity
- secure storage
- key distribution problem

Asymmetric Encryption

- separate keys for encryption and decryption (a ‘key pair’)
- computationally infeasible to derive one from the other
- one key is (can be) public/published; the other is kept private
- therefore subject to many new attacks
- keys must be huge to prevent brute-force attacks, and algorithms must be resistant to chosen-plaintext attacks
- often called ‘public key encryption’ — it is quite safe to publish the encryption key

Encrypting and signing

- Call Alice’s private key $d_A$ and her public key $e_A$. Alice publishes $e_A$ but keeps $d_A$ secret.
- To send a message secretly to Alice, encrypt it with $e_A$; only Alice can read it, which she does using $d_A$.
- If Alice wants to prove she originated a message, she can encrypt it using $d_A$. Then anyone can get hold of $e_A$ and read the message, and also know that it must have been encrypted using $d_A$ — i.e. it was encrypted by Alice.
- If Alice wants to send a secret message to Bob, and have Bob know it came from her, she should first encrypt it with $d_A$, and then with $e_A$.

But

- In fact, asymmetric encryption is computationally expensive, so we wouldn’t do the above in practice.
- to prove Alice sent $m$: create a hash of the message, and sign that; $m_{[h(m)]_{d_A}}$
- to send a secret message to Alice: use a symmetric session key to encrypt the message, placing this at the beginning of the message, encrypted under $e_A$. $[k_{[m]}_{e_A}]$
- How to do signing and encryption together, in practice, then?
Comparison

<table>
<thead>
<tr>
<th>Symmetric Encryption</th>
<th>Asymmetric Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 – 256 bit keys</td>
<td>512 – 4096 bit keys</td>
</tr>
<tr>
<td>one key per two parties communicating</td>
<td>one ‘key pair’ per individual</td>
</tr>
<tr>
<td>efficient, especially in hardware</td>
<td>computationally expensive</td>
</tr>
<tr>
<td>DES, AES, Blowfish, Caesar, Vigenere, Enigma</td>
<td>RSA, El Gamal, Elliptic Curves</td>
</tr>
<tr>
<td>use with ECB, CBC, ...</td>
<td>use sparingly, usually to encrypt other keys, or to sign hashes</td>
</tr>
</tbody>
</table>

Summary

- Cryptography
  - history and concepts
  - perennial issues
- Concepts surrounding cryptography
  - types of cipher
- One-way functions
  - passwords are problematic
  - hashes are similar to ciphers, but typically simpler
- Symmetric encryption and block modes
  - easy to use a good crypto algorithm badly
- Asymmetric Encryption
  - need a clear head; the e's and d's will trip you up eventually
Review: Block Cipher

- Our general goal is to define \( f \)
- A lookup table would be ideal, but impractical
- So \( f \) must be a mathematical function
- We have seen how to use \( f \) in encrypting a whole message
- Now we consider the design of \( f \) itself.

DES: Data Encryption Standard

- 64-bit block cipher
- 56-bit key
  - Often expressed as a 64-bit number with parity checking
- Baroque design
  - Mostly proposed by IBM (based on earlier work 'Lucifer')
  - 'Approved' by NSA;
- Originally designed to run on custom hardware
- Eventually ISO/ANSI standard;
  - Also known as DEA (data encryption algorithm)
- Adopted 1976; ANSI standard 1981;
- NIST endorsement withdrawn, 19th May, 2005.
04 Cryptography in Practice

Security Principles

Initial permutation

- simple re-ordering of bits
  - helps construct the algorithm; not cryptographically significant
- L₀ gets bits numbers:
  - 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8
- R₀ gets bits numbers:
  - 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7

F-function

- 48 bits (as eight 6-bit blocks)
- S-boxes
  - S₀, S₁, S₂, S₃, S₄, S₅, S₆, S₇
- 32 bits
- permutation (P-box)
S-boxes

<table>
<thead>
<tr>
<th>Row No.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>12</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- simple fixed look-up table
  - outermost bits of 6-bit word form row, remainder form column number
- total of eight such tables/boxes

From FIPS PUB 46-3
FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, 1999 October 25

Design Issues

- S-boxes are carefully designed
  - and were tweaked by NSA
- rumours of 'back-doors'
- weak keys exist
- *compliment key property* reduces 'brute force' search space
- clear means for 'avalanche effect'
- same algorithm works in reverse
  - just build the key schedule backwards
- much discussion about key length!

Breaking DES

- Brute Force
  - potentially as a 'known ciphertext attack' - the weakest kind of attack
- By cryptanalysis
  - usually as a 'chosen plaintext attack' - the strongest kind of attack

Key Scheduling

- key expressed as 64-bit value passes through 'permuted choice 1' (not shown) to yield 56 bits of real key material, k
- left shift (rotate) by one or two bits, depending on the round
- permuted choice 2 is sometimes called a 'compression permutation'

Design Issues

- S-boxes are carefully designed
  - and were tweaked by NSA
- rumours of 'back-doors'
- weak keys exist
- *compliment key property* reduces 'brute force' search space
- clear means for 'avalanche effect'
- same algorithm works in reverse
  - just build the key schedule backwards
- much discussion about key length!
Brute force

- Brute force is the simplest attack
  - try every key in turn
- if you know the key generation process, try to re-run it
  - e.g. use keys based on hashes of dictionary words
  - e.g. Ubuntu OpenSSL/random number generator bug
  - otherwise, start at 0x00 0000 0000 0000 and go systematically through to 0xFF FFFF FFFF FFFF (etc.)
- \( n \)-bit key gives rise to \( 2^n \) possible keys
  - comparison: \( 2^{25} \) seconds in a year; \( 2^{50} \) seconds since the big bang
  - following Moore's law gives a factor of up to \( 2^7 \) speed-up in a decade
- massive parallelism helps, of course
  - if we could make quantum computing do this kind of thing, we'd really get somewhere

Chinese Lottery – a hypothetical attack

- suppose you control the means of production...
- suppose you equip every receiver with a DES-breaking chip

<table>
<thead>
<tr>
<th>Country</th>
<th>Population</th>
<th>number of Radios/TVs</th>
<th>time to break</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>56 bit</td>
<td>64 bit</td>
</tr>
<tr>
<td>China</td>
<td>1 190 431 000</td>
<td>257 000 000</td>
<td>280s</td>
</tr>
<tr>
<td>USA</td>
<td>260 714 000</td>
<td>739 000 000</td>
<td>97s</td>
</tr>
<tr>
<td>Iraq</td>
<td>19 890 000</td>
<td>4 730 000</td>
<td>4.2h</td>
</tr>
<tr>
<td>Israel</td>
<td>5 051 000</td>
<td>3 640 000</td>
<td>5.5h</td>
</tr>
<tr>
<td>Wyoming</td>
<td>470 000</td>
<td>1 330 000</td>
<td>15h</td>
</tr>
</tbody>
</table>

What really happened... RSA Labs Challenge

<table>
<thead>
<tr>
<th>Contest</th>
<th>Prize</th>
<th>Start</th>
<th>End</th>
<th>Time for Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>$10,000</td>
<td>28 January 1997, 9 am PST</td>
<td>17 June 1997, 10:40 pm PST</td>
<td>140 days</td>
</tr>
<tr>
<td>RC5-32/12/5</td>
<td>$1,000</td>
<td>28 January 1997, 9 am PST</td>
<td>28 January 1997, 12:30 pm PST</td>
<td>3.5 hours</td>
</tr>
<tr>
<td>RC5-32/12/6</td>
<td>$5,000</td>
<td>26 January 1997, 9 am PST</td>
<td>10 February 1997, 10:00 am PST</td>
<td>313 hours</td>
</tr>
<tr>
<td>RC5-32/12/7</td>
<td>$10,000</td>
<td>28 January 1997, 9 am PST</td>
<td>20 October 1997, 11:18 am PST</td>
<td>265 days</td>
</tr>
<tr>
<td>RC5-32/12/8</td>
<td>$10,000</td>
<td>28 January 1997, 9 am PST</td>
<td>14 July 2002, 01:50 UTC</td>
<td>1757 days</td>
</tr>
</tbody>
</table>

Active search time as reported by winner

http://www.rsa.com/rsalabs/node.asp?id=2100

http://www.rsa.com/rsalabs/node.asp?id=2103

What really happened... RSA Labs Challenge

- Challenge II-1: distributed.net solved in 41 days, 1998
- Challenge II-2: purpose-built machine, 56 hours, July 1998
  - machine cost $250000; prize was $10000.
- “In 1999, the Electronic Frontier Foundation’s "Deep Crack" machine, in combination with distributed.net, successfully solved RSA's DES Challenge III in 22 hours and 15 minutes.”

http://www.rsa.com/rsalabs/node.asp?id=2100
Chosen plaintext attacks

- Two significant techniques developed to attack DES
  - work (with varying degrees of efficiency) against many similar algorithms
- Rely on
  1. analysing the algorithm structure,
  2. computing probabilities that bits of the key are a 1 or a 0,
  3. and then encrypting massive numbers of plaintexts until the accumulated data allows the probabilities to converge (so one key is ‘overwhelmingly likely’ – and easy to check)

Differential cryptanalysis

- look at XOR-difference between pairs of plaintexts and corresponding ciphertexts
- recovers a DES key with, on average $2^{47}$ plaintexts
- if the number of rounds is 17 or 18, becomes about as hard as brute force
- 19 rounds or more, becomes impossible: needs more than $2^{64}$ plaintexts
- published in 1990; transpires that DES’s designers knew the technique — which is why it doesn’t help too much, and why 16 rounds were chosen

Linear cryptanalysis:

- make linear approximations of the block cipher;
- work by joining together 1-round linear approximations;
- some S-boxes are easier to exploit than others
- on average can recover key with $2^{43}$ plaintexts; best known attack against DES
- DES is relatively weak against this attack — either the spooks didn’t know about it, or had some other motive!

Double encryption

- Does encrypting twice
  
  $\text{DES}(k_2,\text{DES}(k_1,\text{plaintext}))$

- halve the security?
- double the security?
- square the security?
- not make much difference?
part of the answer has to do with the mathematical theory of groups:
- Elements: ciphertext blocks
- Binary operation: composition

if these form a group, then two encryptions are no better than one

DES has been shown to be not at all group-like

not the whole story …
- 'meet in the middle attack' makes double encryption theoretically weak

3DES (triple DES): interim solution to DES weakness

- Use three DES encryptions in series
- Effectively 168-bit key size

3DES = DES(K1) ; DES(K2) ; DES(K3)
- Encrypt-Decrypt-Encrypt
  - Can build backward-compatible hardware this way
  - Does this affect the strength?
- Strong but slow

- DES with independent subkeys is also possible;
  - many other DES variants exist: e.g. 'export strength' 40-bit DES

AES: Advanced Encryption Standard

- DES has given rise to decades of research in symmetric cryptography and cryptanalysis
  - many other algorithms along the way: IDEA, RC2, RC5, Fortezza, Blowfish, Twofish, …
- AES is designated successor to DES and 3DES
- Result of open competition
  - and two years' public and private review
- Winner Rijndael
  - Two authors from Belgium (J. Daemen and V. Rijmen)
- Formal standard: Federal Information Processing Standards (FIPS) Publication 197
- Open standard
  - Source code/reference implementations available from day one.

AES: options

- Data blocks of 128 bits
- Key lengths of 128, 192, and 256 bits
- Corresponding number of rounds 10, 12, or 14
- Algorithm allows for other options not endorsed by NIST
- Intended for hardware or software

- So far, seems strong:
  - But doubt cast on the key-scheduling in the 256-bit/14-round version
AES: pseudocode

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1]) // See Sec. 5.1.4
    for round = 1 step 1 to Nr
        SubBytes(state) // See Sec. 5.1.1
        ShiftRows(state) // See Sec. 5.1.2
        MixColumns(state) // See Sec. 5.1.3
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb,(Nr+1)*Nb-1])
    out = state
end

Public-key algorithms

• recall that now we are looking for algorithms which use two keys
  — one for encryption and one for decryption
• so the keys must be related
  — but we don’t want it to be possible to derive one if you know the other
• this allows us to make one key public, and keep the other secret
• so the algorithms are quite different
  — rely on ‘hard’ maths problems, i.e., no efficient (non-quantum) algorithms known
  — however, verifying the solution is simple
• various parts of mathematics have been proposed for this purpose; leading solutions are
  — RSA (factorisation)
  — Diffie-Hellman (discrete logarithm)
  — Elliptic Curve Diffie-Hellman (Elliptic curves)

Some difficult problems: factorisation

• Integer factorisation
  — Finding prime numbers of an composite number
  — Example: 15 = 3 x 5, where 3 and 5 are prime numbers
  — RSA-768 composite number:
  123018664531657230585370333981369938514678507231631789671451056394856573557195553288960929024507087509674383671609642387467908720370223082564329699624808350897373604968947881108722514044774453853352081803773875020538635236522018694330566234325065393493823917164378887045627898266522441850588038707539905561822516173055532094337126381093609920361062957091703490626578513654002258120099348739432864386492702689442137150581603533654896852727976906963052626547781001966320826242622715067675906036889723152162276180576687232448928128490314254413557196514234711
  — Factorisation of RSA-768 took 2 years using hundreds of machines
  — In contrast to factorisation, checking the solution is easy (just multiply the prime numbers and see if you get the composite)

Some difficult problems: discrete logarithm

• (Ordinary) logarithm problem
  — \( \log_b a \) find an exponent \( x \), such that \( b^x = a \)
• Discrete logarithm problem
  — \( g^x \equiv c \mod p \)
  — \( x \) is called the discrete logarithm of \( c \) modulo \( p \) to the base \( g \)
  — Example: \( 2^3 \equiv 1 \mod 5 \)
  — is the discrete logarithm of 1 modulo 3 to the base 2
  — The calculation of the discrete logarithm \( z \) when given \( a, c, \) and \( p \) is a computationally difficult problem and the asymptotical runtime of the best known algorithms for this problem is exponential in the bitlength of \( p \)
  — Verifying the discrete logarithm (called discrete exponentiation) is not difficult
RSA algorithm

- named for Rivest, Shamir, and Adelman
  - published the algorithm in 1978
- was in fact previously discovered – in secret
  - by Clifford Cocks at GCHQ (CESG), in 1973
- based on modular arithmetic

RSA – the main features

- Set-up
  1. choose prime numbers \( p \) and \( q \).
  2. compute \( n = p \times q \)
  3. select \( d \) and \( e \), such that
    i. \( d \) is relatively prime to \( (p-1)(q-1) \), and
    ii. \( (e\times d) \mod (p-1)(q-1) = 1 \)
  4. discard \( p \) and \( q \)
  5. public key is the pair \((e,n)\) and private key is the pair \((d,n)\)
- operation (plaintext \( P \), ciphertext \( C \))
  - encrypt: \( C = P^e \mod n \)
  - decrypt: \( P = C^d \mod n \)

RSA issues

- large prime numbers?
  - fine: use a probabilistic prime checker (Rabin-Solway-Strassen)
- \( d \) can be selected by making it any prime larger than both \( p \) and \( q \)
- find \( e \) using Euclid’s algorithm — polynomial time
- exponentiation mod \( n \) can also be done in polynomial time
- you can bias the speed by choosing \( e \) to be small, say — makes encryption faster, decryption slower; popular choice are 3, 17, 65537 (choosing \( e = 3 \) is vulnerable to the Low-exponent attack)
- in software, typically 100 times slower than DES; in hardware, about 1 000 times slower

Breaking RSA

A. Factoring \( n \).

  History suggests this is a hard problem.
  
  if you can find \( p \) and \( q \), then knowing \( e \), you can easily find \( d \)
B. Finding \( (p-1)(q-1) \) without factoring \( n \).

  This is arguably about as hard.
**ECC: Elliptic Curve Cryptography**

- Another basis for asymmetric (public key) cryptography
- Desirable because it has better scaling properties than RSA (much shorter keys)
- Elliptic curve cryptography (ECC) was proposed independently by Victor Miller and Neal Koblitz in 1985/1987.
- Rather more complex mathematically
  - many implementations
- Becoming widely adopted

**Elliptic Curve – the main features**

- Elliptic Curve $E$ over Real Numbers ($\mathbb{R}$)
  - $y^2 = f(x)$ for a cubic polynomial $f(x)$
  - E.g.: $E(3,18)$: $y^2 = x^3 - 3x + 18$
- $E(a,b) = \{(x,y) : y^2 = x^3 + ax + b \} \cup \{O\}$
  - set of all points on elliptic curve and an extra point $O$ at “infinity”
  - defines an abelian group, provided that discriminant $D \neq 0$
  - i.e., $4a^3 + 27b^2 \neq 0$

**Elliptic Curve – group law**

- Point "O" (Point of infinity) Lies on every vertical line Serves as additive identity
  - $O + O = O$
- Inverse
  - $P = (x_1,y_1)$
  - $-P = (x_1,-y_1)$
  - $P + (-P) = O$

**Elliptic Curve – addition**

- Addition of two points
  - $P = (x_1,y_1)$ and $Q = (x_2,y_2)$
  - $P + Q = (x_3,y_3)$
- Algebraic
  - $x_3 = \Delta^2 - x_1 - x_2$
  - $y_3 = \Delta(x_1 - x_3) - y_1$
  - $\Delta = \begin{cases} 
    \frac{(y_2 - y_1)(x_2 - x_1)}{x_2 - x_1}, & \text{where } P \neq Q \\
    3x_1^2 + a, & \text{where } P = Q
  \end{cases}$
Elliptic Curve over $\mathbb{Z}_p$ (Prime Curves)

- Similar to EC over $\mathbb{R}$.
- $\mathbb{Z}$ is the set of integers, $\mathbb{Z}_p = \{0,1,2,..., p-1\}$, where $p$ is a prime number.
- Variables and coefficients are restricted to elements of $\mathbb{Z}_p$.
- $E = \{(x,y) : y^2 \mod p = (x^3 + ax + b) \mod p \} \cup \{O\}$, where $x,y,a,b \in \mathbb{Z}_p$.
- Defined an abelian group, provided that discriminant $D \neq 0$.
- $4a^3 + 27b^2 \mod p \neq 0$.
- Example: $a=1, b=1, x=9, y=7, p=23$.
  - $7^2 \mod 23 = (9^2 + 1) \mod 23$
  - $49 \mod 23 = 196 \mod 23$
  - $3 \times 3 = (9,7) \in E_p(1,1)$
- Addition remains the same, but also within modular operation.

Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

- Choose
  - $E_p(a,b)$: large prime $p$ and EC parameters $a,b$.
  - Point $G = (x_G,y_G) \in E_p(a,b)$
    - Order of $G$ should be very large value.
    - The order $n$ of a point $G$ on an elliptic curve is the smallest positive integer $n$ such that $G \times n = O$.
- ECDH Public parameters: $E_p(a,b)$ and $G$.

Alice

<table>
<thead>
<tr>
<th>Private Key: $n_a &lt; n$</th>
<th>$P_a$</th>
<th>$P_b$</th>
<th>$K = P_a \times n_b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Key: $P_a = G \times n_a$</td>
<td>Public Key: $P_b = G \times n_b$</td>
<td>Shared Key: Key = $P_a \times n_b$</td>
<td></td>
</tr>
</tbody>
</table>

Bob

$P_a \times n_b = (G \times n_a) \times n_b = (G \times n_a) \times n_b = P_a \times n_b$

Efficiency of ECC

- NIST recommended key sizes (in bits):

<table>
<thead>
<tr>
<th>Symmetric Key Size</th>
<th>RSA and DH Key Size</th>
<th>EC Key Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1024</td>
<td>160</td>
</tr>
<tr>
<td>112</td>
<td>2048</td>
<td>224</td>
</tr>
<tr>
<td>128</td>
<td>3072</td>
<td>256</td>
</tr>
<tr>
<td>192</td>
<td>7680</td>
<td>384</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>521</td>
</tr>
</tbody>
</table>

- E.g.: To protect a symmetric 128-bit AES key one should use a 3072-bit RSA key or a 256-bit ECDH key.

Crypto Performance: symmetric algorithms

- Benchmarks using OpenSSL (using 1 of the 4 cores), Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>16 bytes</th>
<th>64 bytes</th>
<th>256 bytes</th>
<th>1024 bytes</th>
<th>8192 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc4</td>
<td>413638.26k</td>
<td>641908.25k</td>
<td>737861.80k</td>
<td>778667.72k</td>
<td>775034.20k</td>
</tr>
<tr>
<td>des</td>
<td>62096.76k</td>
<td>63910.98k</td>
<td>63973.37k</td>
<td>63590.06k</td>
<td>63946.75k</td>
</tr>
<tr>
<td>aes</td>
<td>38613.41k</td>
<td>39182.72k</td>
<td>39425.71k</td>
<td>39482.37k</td>
<td>39531.86k</td>
</tr>
<tr>
<td>blowfish</td>
<td>100321.62k</td>
<td>111338.43k</td>
<td>112307.80k</td>
<td>112809.64k</td>
<td>113589.34k</td>
</tr>
<tr>
<td>cast</td>
<td>98649.05k</td>
<td>102900.35k</td>
<td>104184.83k</td>
<td>104588.97k</td>
<td>104445.27k</td>
</tr>
<tr>
<td>aes-128</td>
<td>100340.98k</td>
<td>108169.22k</td>
<td>109688.27k</td>
<td>110206.63k</td>
<td>111353.86k</td>
</tr>
<tr>
<td>aes-192</td>
<td>86331.45k</td>
<td>90609.98k</td>
<td>92677.97k</td>
<td>93344.19k</td>
<td>93025.62k</td>
</tr>
<tr>
<td>aes-256</td>
<td>78056.76k</td>
<td>78227.01k</td>
<td>79286.70k</td>
<td>80326.31k</td>
<td>80153.26k</td>
</tr>
</tbody>
</table>

Crypto Performance: asymmetric algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>sign</th>
<th>verify</th>
<th>sign kb/s</th>
<th>verify kb/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsa</td>
<td>512 bits</td>
<td>0.000053s</td>
<td>0.000049s</td>
<td>18866.8</td>
</tr>
<tr>
<td>rsa</td>
<td>1024 bits</td>
<td>0.000184s</td>
<td>0.000012s</td>
<td>5425.6</td>
</tr>
<tr>
<td>rsa</td>
<td>2048 bits</td>
<td>0.001324s</td>
<td>0.000041s</td>
<td>755.1</td>
</tr>
<tr>
<td>rsa</td>
<td>4096 bits</td>
<td>0.009569s</td>
<td>0.000153s</td>
<td>104.5</td>
</tr>
</tbody>
</table>

Crypto Performance: hashing algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>16 bytes</th>
<th>64 bytes</th>
<th>256 bytes</th>
<th>1024 bytes</th>
<th>8192 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>md2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mdc2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>md4</td>
<td>81569.65k</td>
<td>245852.74k</td>
<td>567837.18k</td>
<td>843165.01k</td>
<td>983283.84k</td>
</tr>
<tr>
<td>md5</td>
<td>58067.51k</td>
<td>169312.94k</td>
<td>373451.18k</td>
<td>535094.95k</td>
<td>612229.12k</td>
</tr>
<tr>
<td>hmac(md5)</td>
<td>46428.89k</td>
<td>145214.39k</td>
<td>335519.15k</td>
<td>509540.35k</td>
<td>606142.46k</td>
</tr>
<tr>
<td>sha1</td>
<td>63343.55k</td>
<td>180989.91k</td>
<td>396493.40k</td>
<td>555289.60k</td>
<td>635505.32k</td>
</tr>
</tbody>
</table>

Crypto Performance: asymmetric algorithms (ECC)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>sign</th>
<th>verify</th>
<th>sign kb/s</th>
<th>verify kb/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecdsa</td>
<td>160 bits</td>
<td>0.0001s</td>
<td>0.0002s</td>
<td>15766.4</td>
</tr>
<tr>
<td>ecdsa</td>
<td>192 bits</td>
<td>0.0001s</td>
<td>0.0003s</td>
<td>13140.8</td>
</tr>
<tr>
<td>ecdsa</td>
<td>224 bits</td>
<td>0.0001s</td>
<td>0.0004s</td>
<td>10463.3</td>
</tr>
<tr>
<td>ecdsa</td>
<td>256 bits</td>
<td>0.0001s</td>
<td>0.0004s</td>
<td>9015.5</td>
</tr>
<tr>
<td>ecdsa</td>
<td>384 bits</td>
<td>0.0002s</td>
<td>0.0009s</td>
<td>4667.3</td>
</tr>
<tr>
<td>ecdsa</td>
<td>521 bits</td>
<td>0.0004s</td>
<td>0.0020s</td>
<td>2482.5</td>
</tr>
</tbody>
</table>

ecdsa = elliptic curve digital signature algorithm
Cryptography in Practice

Crypto Performance: summary

- Relative Computation Costs of Diffie-Hellman and Elliptic Curves

<table>
<thead>
<tr>
<th>Security Level (bits)</th>
<th>Ratio of DH Cost : EC Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>3:1</td>
</tr>
<tr>
<td>112</td>
<td>6:1</td>
</tr>
<tr>
<td>128</td>
<td>10:1</td>
</tr>
<tr>
<td>192</td>
<td>32:1</td>
</tr>
<tr>
<td>256</td>
<td>64:1</td>
</tr>
</tbody>
</table>


Side-note

- More ‘exotic’ asymmetric cryptography also exists.
- For example, have multiple encryption keys, and a single decryption key — or vice versa
- sometimes used for ‘group signatures’
- useful for preserving privacy or anonymity
- details are out of scope for us

Digital Signatures

- involve cryptography
- can include timestamps
- usually sign a message digest
- use the properties of asymmetric cryptography
- encrypt with private key $[m]_{DA}$
- anyone can verify (decrypt) with public key, $eA$
- desirable signature properties:
  - authenticity — the signer deliberately signed
  - unforgeability
  - not re-usable
  - document unalterable after signature
  - cannot be repudiated
  - none of these is entirely true of pen & paper signatures

Practicalities

- Asymmetric algorithms are too inefficient to do this in practice
- Sign a hash of the message, not the message itself
  $m[h(m)]_{DA}$
- key compromise is an inherent problem
  — how do you (as a relying party) distinguish
  — from
  sign $\rightarrow$ compromise $\rightarrow$ repudiate (deliberate compromise)
  sign $\rightarrow$ compromise $\rightarrow$ repudiate (accidental compromise)
- talk of Alice or Bob is misleading
  — the signature is created by software not by the person
  — whether or not the signature is applied to the data Alice expects is entirely in the hands of the interface designer
Hash Algorithms in use

**MD5**
- processes input text in 512-bit blocks
- output is four 32-bit blocks; concatenate to a 128-bit hash value
- multiple ‘rounds’ like DES etc., based on bitwise, AND, OR, NOT, and left circular shift
- several groups have demonstrated serious, repeatable, hash collisions
- use of MD5 discouraged — but not dead yet

**SHA-1**
- processes input text in 512-bit blocks
- similar construction, but uses five 32-bit blocks instead of four; thus giving a 160-bit output
- some mystery in the design; NSA had a hand in it
- SHA-1 is very widely used, but looking shaky: increasingly good attacks are being found; consensus is that it needs replacing.

**SHA-2**
- ‘Next generation’, sharing some details with SHA-1
- most popular variant, SHA-256 is gaining ground
- commonality with SHA-1 makes cryptographers uneasy

**SHA-3**
- open competition in progress to find a fresh replacement
- round one: 64 entrants, November 2008
- round two: 14 remaining candidates; started July 2009
- round three: 5 finalists announced December 2010
- final result (winner) due mid-2012

---

**Hash Algorithms**

**SHA-2**
- ‘Next generation’, sharing some details with SHA-1
- most popular variant, SHA-256 is gaining ground
- commonality with SHA-1 makes cryptographers uneasy

**SHA-3**
- open competition in progress to find a fresh replacement
- round one: 64 entrants, November 2008
- round two: 14 remaining candidates; started July 2009
- round three: 5 finalists announced December 2010
- final result (winner) due mid-2012


**Attacks on Hashing and Signatures**

- ideal attack:
  1. find two messages that hash to the same digest
  2. one is nice; one nasty
  3. get Alice to sign the digest, seeing that it corresponds to the nice message
  4. distribute the nasty one, and say that Alice signed it
  5. everyone can verify that this is true
- Step (1) is difficult if you are given one of the messages by Alice
  — see following slides; compare the ‘birthday book paradox’
**Attack One: Given Message**

- Alice’s document
- \( n \)-bit digest of Alice’s document
- 2\(^n\) possible messages
- look for a match
- candidate digest

**Attack two: choosing both messages**

- 2\(^{n/2}\) possible messages
- look for a match
- candidate digest

---

**nice...**

John’s assignment was an outstanding piece of work. He has demonstrated a clear understanding of the subject, and has outclassed his fellow-students. The answer to Question 4 was particularly impressive, in that it used a technique that is far above any of the usual methods. This is a significant breakthrough, and should receive publication as soon as possible.

**nasty...**

The assignment submitted by John was most disappointing. He completely misunderstood the questions, and demonstrated very little understanding of the subject. He is clearly the weakest student in this class. A clear demonstration of his shortcomings is seen in the answer to Question 4. The work presented is at odds with all of the course material and literature, and cannot possibly be right.
04 Cryptography in Practice

**Outcome**

- $2^{16}$ possible versions of each message
- With a (hypothetical) 32 bit hash, this gives a very good chance of finding two messages with the same hash.
- If no hash collisions are found, re-run with a few more options.
- Examiner Alice signs the nice one; nasty one is put on file
- John Smith is well and truly stitched-up
- The moral of the story: your hash needs to be twice as long as you thought

**Documented Attacks on MD5**

- 1996: collision attacks identified
- 2005: researchers release pair of PostScript documents which render to different texts but have identical hashes — and something similar for digital certificates
- 2008: fairly comprehensive attack against MD5-signed digital certificates

**Quantum Computing**

- Oxford Centre for Quantum Computation is a good source of information
- [http://www.qubit.org/tutorials](http://www.qubit.org/tutorials)
- by using quantum effects, create a qubit register which holds both values 1 and 0 simultaneously — call this ‘superposition’
- put together n qubits to build a register holding $2^n$ values simultaneously
- can think of this as a collection of probability coefficients — must add up to 1.
- apply an operation to the register — change the coefficients
- making an ‘observation’ destroys the superposition, and delivers a single answer
- with the ‘right’ operations, the desired answer becomes overwhelmingly likely

**Quantum Computing Practicalities**

- Shor’s Algorithm
  - rapidly factorizes (large) numbers
  - equivalently enables us to reverse discrete logarithms
  - result: easy break of asymmetric cryptography
  - state-of-the-art appears to be 10 years old
    - (15 = 5 x 3 computed in 2001 at IBM)
- Grover’s Algorithm
  - rapidly search unsorted data
  - equivalently, invert a non-invertable function
Quantum Cryptography

- uses similar effects to quantum computing, but in a different – and so far more successful – way.
- claimed as a solution to the key distribution problem
- eavesdropping is detectable according to the laws of physics
  — no passive observers exist at the quantum level (c.f. Heisenburg)
- quantum cryptography devices are available to buy
  — apparently work over many km today

### Example implementation

1. Photons polarized at 0, 45, 90, or 135 degrees
2. Recipient can measure polarization, either the rectilinear or the diagonal, but not both for a single photon
3. Sender sends photons, choosing polarizations at random
4. Recipient chooses detection mode at random
5. Recipient publishes the detection mode chosen
6. Where it was the wrong mode, both parties throw away the bit.
7. Where it was the right mode, they have an un-eavesdropped (sequence of) bit(s)
8. Eavesdropper would need to detect and retransmit photons – statistically impossible to get right.

### Summary

- Good, commercial-grade cryptography is readily available
- Implementations can be made quite efficient
  — but not without resource costs
- Asymmetric Algorithms are all much less efficient than symmetric ones
  — but are adequate for signing hashes, encrypting session keys, etc.
- Hashing is a bit of a mess right now
  — but is heading for a good outcome
- Quantum cryptography provides alternatives for establishing encrypted channels
  — few obvious use cases
- Quantum computing might some day defeat all asymmetric algorithms