Cryptographic Algorithms Revealed

Greg Rose
ggr@qualcomm.com

Copyright 2000 Greg Rose. Permission is granted to make one printed copy from the electronic version of this material. The printed notes may not be reproduced. No other reproduction may be made without express permission.

Greg Rose, QUALCOMM Australia

Greg Rose graduated from the University of New South Wales with a B.Sc. (honours) in computer science and was awarded the University Medal in 1977. A member of the Board of Directors of the USENIX Association, he served as program chair of the 1996 USENIX Security Symposium. As Principal Engineer at QUALCOMM, he focuses on cryptographic security and authentication for wireless communications, and on setting up the office of QUALCOMM Australia. He has written a number of public tools using cryptography, and he holds generic cryptographic export licenses for two countries.

Overview

- Brief history
- Introductory topics
- Symmetric Block ciphers
- Block cipher cryptanalysis
- Symmetric Stream ciphers
- Public Key systems
- Hash functions

I feel it is always enlightening to see some of the history behind a subject.

There is much more to cryptography than algorithms, such as key management, protocols, and so on.This tutorial does not attempt to address these other things.

Historical perspective

- "secret writing" goes back a long way
  - 1900 B.C. -- non-standard hieroglyphics
  - 500 B.C. -- "atbash"
  - Julius Caesar and *rot3*
- Used steadily through the middle ages
  - monarchies
  - religious groups
- Lovers' notes around the turn of the century
- *Never as secure as they thought!*

atbash was a Hebrew cipher in which the first letter of the Hebrew alphabet (aleph) was replaced with the last (taw), and so on. It is self-inverse like rot13.

George Bernard Shaw and Oscar Wilde would supposedly translate mono-alphabetic substitution ciphers (like newspaper "cryptoquotes") in their heads… and have a good laugh.

More history

- Modern theory starts around the U.S. Civil War
  - Playfair
  - Thomas Jefferson's wheel
- Extensive use of code books
  - Telegrams and commercial codes
  - Vernam cipher
  - World War I
  - U.S. Prohibition era

In "Jefferson's Wheel" there were a number of disks on a single axle, with random alphabets around them. To encrypt a message, the disks were lined up so that the letters formed a line. Then any other line of letters was the ciphertext. The "key" was the alphabets on the disks, and the order they were put on the axle.

The "Vernam cipher" is better known these days as a "One Time Pad".

Modern history

- World War II
  - Enigma, Bletchley Park, Colossus
  - Japan's PURPLE
- Claude Shannon and Information Theory
- Cold war, everything went quiet. NSA formed.
- 1974, public interest resumes
  - Data Encryption Standard (DES)
  - "New Directions in Cryptography"
    - Diffie and Hellman's introduction of Public Key Cryptography
  - RSA (Rivest, Shamir, Adelman)

This is a good place to mention David Kahn's excellent book "the Codebreakers: the Story of Secret Writing", Macmillan 1967, recently reprinted with minor additions. Also James Bamford's "the Puzzle Palace", about the National Security Agency.

Really modern history

- DES was supposed to be replaced in '89, and '94, but was recertified both times
- In '98 the DES cracking engine was built by EFF
- In '98 the effort to develop the Advanced Encryption Standard was started.
  - 15 candidates
  - 5 broken, 5 not as good as others
  - 5 left
  - Rijndael (Rijmen & Daemen) selected

"*Cracking DES*", Electronic Frontier Foundation.

Http://aes.nist.gov for all about all the AES candidates. I won't bother giving individual references.

Cryptosystems, Key Management, and Hard Stuff

- What is a cryptosystem?
- What are keys?
- Why do we have to manage them?
- Why is managing them hard?

Cryptosystems

- Nothing to do with *SEX!*
- Everything to do with security.
- A *cryptosystem* is a cryptographic algorithm,
  - +the key or password management
  - +the environment
  - +the network
  - +the protocol
  - +the people
  - +everything else

In case you hadn't guessed, the SEX was to make you pay attention.

Sometimes cryptosystems are referred to as Cryptographic Protocols, but I find this confusing and sometimes too restrictive.

Key (Cryptovariable) Management

- All secrecy should reside in the keys
  - sometimes there are reasons for keeping secret other aspects, but not for crypto algorithms.
- Many tradeoffs:
  - long term vs. short term
  - communications vs. storage
  - secure vs. easy to remember
  - personal vs. corporate vs. recoverable
- Keep them secret!
- Remember them!

17-May-01 Copyright Greg Rose, 2000-2001 slide 9

That "all secrecy should reside in the key" is *Kerckhoff's Maxim*.

The best way to decide among the tradeoffs is by understanding the threats. More on this later.

When keeping something secret, assign a value to what is being protected, and protect the key "that much". As the thing gets more valuable, use a better key. DES encryption is probably fine for most of our love letters (perhaps not Bill Clinton's though).

Then again, stronger keys don't cost much (except to manage them, or export them).

When keeping something secret, assign a cost to losing it… then make sure you put that amount of effort into remembering the key. Corporate key recovery really does make some sense.

Entropy

- A mathematical term
- Measures "the actual amount of information"
- English sentences have about 1.5 bits per character
  - therefore, a passphrase for a 128 bit key would be about 80 characters long!
- Relates to "predictability" and so is relevant to security
  - you have no security if your secret can be guessed
- Good practice to compress first

17-May-01 Copyright Greg Rose, 2000-2001 slide 10

I'm avoiding too much mathematics, but the actual formula is pretty simple. If there is a set of *N* possibilities, and one of them must occur, let *p(i)* be the probability of the *i*th one. Then the entropy is:

$$? -p(i)\log_2(p(i))$$

It turns out that this (rounded up) is the minimum number of bits which can represent all of the possibilities.

There are other definitions of *entropy*, but this is the one that is meant in this tutorial.

A great reference for this stuff is *"Basic Concepts in Information Theory and Coding"* (subtitled "the Adventures of Secret Agent 00111") by Golomb, Peile, and Scholtz. It is surprisingly readable.

Random musings

- Random information is often important to cryptographic processes
- Cryptographers mean something special:
  - unpredictable, no matter what else you know
  - The entropy of a random number is the number of bits in the number.
- Sometimes "Pseudo-random" is good enough
  - entropy is (at most) that of the seed value
  - therefore, need *very* good seeds (common error)
- Terms: RNG, PRNG , CSPRNG

17-May-01 Copyright Greg Rose, 2000-2001 slide 11

RNG means "Random Number Generator"; for example, radioactive decay, thermal noise, rolling dice.

PRNG means "Pseudo Random Number Generator"; this is where some algorithm generates numbers with nice statistical properties. Not necessarily good enough for cryptographic applications, though… this says nothing about being unpredictable.

CSPRNG means "Cryptographically Secure PRNG"; this is some sequence of algorithmically generated numbers, but where predicting the next one involves solving a hard problem.

Linear Feedback Shift Registers and Linear Congruential Generators are *not* secure.

Knuth's "Seminumerical Algorithms" is an excellent reference.

The "Birthday Paradox"

- How many people need to be in a room, before you are likely to win the bet "two people have the same birthday"?
- Probability that one has Jan 1 is about 1/365
- But how many *pairs* of people are there?
  - n*(n-1)/2
  - probability that Bob has same birthday as Alice is *still* about 1/365
  - with 20 people, there are 190 *pairs of people*
  - about 50% chance two will share a birthday

17-May-01 Copyright Greg Rose, 2000-2001 slide 12

The birthday paradox is easily explained but widely misunderstood. There are 365 days in a year, so you must need hundreds of people in a room before two of them have the same birthday, right? Wrong! Say there are 20 people in the room. There are 20*19/2 == 190 *pairs* of people, and about a 1/365 chance that the second one has the same birthday as the first one in each pair… so the probability is around 50%. Easy to win bets at parties. (What I can't understand is why it is called a paradox.)

So, for *n* bit hashes, when you get about 2^(n/2)

of them, you have a good chance of a collision. For 64 bits of strength, you need 128 bit hashes.

Note that this rule often applies whenever coincidences can ruin your whole day. For things like message authentication codes, where coincidences don't help, 8 bits is worth a dollar.

Symmetric Keys

- Also called “classical”
- The key used is the same at both ends
  - (or decryption key can be derived from encryption)
- How many people can keep a secret?
- Key management problems
  - either n^2 keys for n people,
  - or $n+1$ and a trusted third party (e.g.. Kerberos)
- “signatures” can be repudiated
- Algorithms are generally fast

17-May-01 Copyright Greg Rose, 2000-2001 slide 13

Question: How many people can keep a secret?
Answer: n, if $n-1$ of them are dead.

No-one uses symmetric algorithms for actual digital signatures, but the corresponding thing is called a “Message Authentication Code”. It has most of the properties of a digital signature except that the person at the other end can forge it.

MACs are often encrypted hashes (bad idea), or the leftovers after encryption steps, or hashes including secret information (effectively a key) in the input. There are good ways to do this, and there are bad ones. See Krawczyk et. al, RFC2104, “HMAC: Keyed hashing for Message Authentication” for a good one.

Public keys

- Also called “asymmetric”
- Keys come in pairs; keep one half secret
  - can’t derive the secret one from the public one
- Solves the key distribution problem… just publish the public keys
- Replaces it with the authentication problem
  - How do you know that the key belongs to who you think it does? Still a research problem.
- Can do digital signatures
- Algorithms slow, keys large

17-May-01 Copyright Greg Rose, 2000-2001 slide 14

When you figure out the authentication problem, it gets replaced by the revocation problem… what happens if you lose control of your private key, and want to stop people using it? I call this “stepping on cockroaches”. “*It’s turtles, all the way down.*” -- Keith Bostic (or maybe Bertrand Russell).

Most public key algorithms are mathematically based. The well accepted ones are usually based on the difficulty of factoring or calculating discrete logarithms in finite fields or groups. Choosing your group carefully (elliptic curves) can speed things up and shrink the keys, but still nowhere near symmetric cryptography numbers.

Hybrid systems

- Often use a combination of Public, Classical and no-key cryptography.
- e.g.. SSL, SSH, PGP.
  - Public keys used for authentication, key exchange
  - Hash and public key for digital signature
  - Dynamic session key and classical cipher for security
  - Random numbers for all sorts of things.
- The more elements, the more places the cryptosystem can fail.

17-May-01 Copyright Greg Rose, 2000-2001 slide 15

Obscurity is not Security

- All security should reside in the keys
- Secret algorithms could hide secret bugs, or secret trapdoors (or public license fees)
- “Keyless” algorithms don’t exist…
  - algorithm is the key
  - input data is the key (autokey cipher)
  - both usually very weak
- claims that something “hasn’t been broken” are meaningless but hard to refute
  - unless “… by Shamir, Coppersmith, …”

17-May-01 Copyright Greg Rose, 2000-2001 slide 16

The assumption that all about the system is known, except for the keys, is “Kerckhoff’s Maxim”, dating back about a century.

Algorithms have to run on the computers, so there is plenty of opportunity to reverse-engineer them, whereas keys can be stored offline, protected, etc.

Autokey ciphers, even if strong, require synchronisation; an attacker can force loss of sync, and usually compromise the cryptosystem. Usually weak against chosen plaintext.

I chose two cryptographers at random above. The list is relatively short (a hundred or two people publicly known).

There is almost no chance that “you” can invent a secure cipher algorithm. Actually, you probably could after enough study, but the hard part is making a secure cipher that is efficient enough to be interesting. There are good ones out there. Choose one. (Trying to can be fun, just don’t make the mistake of thinking it is secure.)

Linearity (“Affine”)

- When a function can be expressed as
  f(x) = *ax* + *b*
- begging the question “what are addition and multiplication?”
- Sometimes linearity can be bad
  - systems of linear equations can be efficiently solved
- “non-linearity” is a measure of how different a function is from nearest affine function.
- An “S-Box” is a convenient implementation

17-May-01 Copyright Greg Rose, 2000-2001 slide 17

To be technical, *linear* means *b* == 0, and *affine* means *b* != 0. The difference is usually irrelevant.

Nonlinearity can me measured using the Walsh-Hadamard transformation, which is sort of related to Fourier transformations. For an excellent online reference to this subject, see Terry Ritter’s web pages

http://www.io.com/~ritter

Because the more non-linear a function is, the more likely it is to be hard to calculate, quite often a non-linear function is implemented as a table lookup. This is true whether the table actually has a mathematical description (e.g. as in Rijndael) or when it has been generated “randomly” (e.g. as in DES, SOBER, or MARS). In this case the table is called an “S-Box” (substitution box) because that’s what DES called it. Note, also, that they’re often called S-Boxes even when not used for substitution as such (e.g. SOBER).

The magic of XOR

- XOR (?) is simply the addition of individual bits, modulo 2.
  - 0 ? 0 = 0
  - 0 ? 1 = 1
  - 1 ? 0 = 1
  - 1 ? 1 = 0
- XOR is self-inverse
- XOR is linear
- XOR is implemented on computers
- XOR is correlated to addition

17-May-01 Copyright Greg Rose, 2000-2001 slide 18

There is more about the real mathematics of the XOR operation coming later, when we discuss fields and rings. XOR is the addition operation for Galois Fields of characteristic 2.

Esoteric things

- Zero knowledge proofs
  - prove that you know a secret without divulging it
- Secret splitting
  - give *n* people part of a secret so that *k* of them, together, know all of it
- Anonymity
- Secret voting
- Encrypted key agreement
  - authenticate and establish a shared key at the same time

17-May-01 Copyright Greg Rose, 2000-2001 slide 19

“Applied Cryptography” 2nd edition, Wiley 1996, by Bruce Schneier, has a very comprehensive list of the weird and wonderful things which can be done with cryptographic techniques.

Some common mistakes

- Too little entropy in the random number seed
  - This was Netscape’s problem
- Using the wrong block cipher mode
- Reusing keys for stream ciphers, or using a block cipher key for too long
  - Microsoft reuses streams in a number of places
- Authenticating at the beginning of a session, but allowing hijacking later
- Choosing bad passwords
- Divide and conquer attacks

17-May-01 Copyright Greg Rose, 2000-2001 slide 20

Netscape seeded their random number generator with the time and the process-id, then used it to generate an encryption key. The time was measurable within a few seconds, and the process-id could only be in a limited range, so by trying about 120,000 starting values and duplicating the computation, the privacy key was easily guessed (Ian Goldberg, U.C. Berkeley)

A banking system used ECB mode for transactions, and the amount field was encrypted independently. By making a random change after the validation was done, a random (but on average very large) amount of money was transferred.

Stored passwords on MS Windows systems were encrypted with the same stream cipher output, so XORing X’s password and encrypted password into Y’s encrypted password returned Y’s real password.

Divide and conquer: solving part of the problem, then going on to solve the rest; the complexity is the sum of the parts, not the product. E.g., a long password was too big for the hashing algorithm, so the halves were hashed independently. I saw yet another example of this in May 2001.

Block Ciphers

- Intro
- General structures
- Block cipher cryptanalysis
- DES in more detail
- Some AES candidates
- AES (Rijndael) in more detail
- Modes of operation

17-May-01 Copyright Greg Rose, 2000-2001 slide 21

Block Ciphers

- Work by taking blocks of input (typically 64 or 128 bits), encrypting whole block
- You can:
  - reuse keys
  - use different *modes* for different purposes
  - have some level of binding and integrity for free
- But you have to be careful
  - block padding and initialisation vectors
  - codebook attacks, use the right modes
- E.g.: DES, Blowfish, IDEA, SAFER, CAST, AES

17-May-01 Copyright Greg Rose, 2000-2001 slide 22

You can only encrypt whole blocks, and you get whole blocks back. So an odd length chunk of data gets bigger. There are various padding techniques to recover the original input. You usually want to use an Initialisation vector too (see later) so it gets bigger still.

All of the above are free, except IDEA, and it is cheap.

DES is the Data Encryption Standard algorithm. It has a short key (56 bits) and basically it has run out of time. There was an effort called the Advanced Encryption Standard, run by NIST, to find a replacement.

You can do multiple encryption for added security, e.g.. triple-DES. Or you can just use an algorithm which accepts a longer key, as all the others do.

Advanced Encryption Standard

- 5 finalists announced in August 1999
  - Serpent (Anderson, Biham, Knudsen)
  - Rijndael (Joan Daemen, Vincent Rijmen)
  - Twofish (Counterpane)
  - Mars (IBM)
  - RC6 (RSA Data Security Inc.)
- 128 bit blocksize
- 128, 192, 256 bit keys
- Rijndael won in late 2000.

17-May-01 Copyright Greg Rose, 2000-2001 slide 23

More detail coming, just hang in there.

General structures

- Substitution-Permutation Networks
  - eg. IDEA, Rijndael
  - use reversible operations to mix and mingle
- Feistel ciphers
  - eg DES (type 1), MARS
  - various “types”
  - use irreversible operations in a reversible structure
    - split into parts
    - leave part unchanged, use a function of that part to affect other part
- Boundaries not so clear any more...

17-May-01 Copyright Greg Rose, 2000-2001 slide 24

Horst Feistel worked at IBM research, and designed Lucifer, the prototype Feistel cipher, from which DES evolved.

Newer designs tend to blend some of the characteristics of both these techniques, but parts will be “S-P like” or “Feistel like”.

Feistel structure

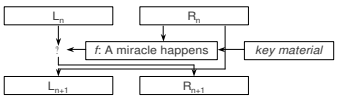
- Multiple "rounds"
  - each round increases the confusion and removes structure
  - reversibility comes from the structure, not the function
- An important property is "avalanche"
  - how many bits of output depend on each bit of input/key
  - once complete, means any change of input should result (on average) in a change in half output bits
  - not provably necessary, provably not sufficient.

17-May-01 Copyright Greg Rose, 2000-2001 slide 25

Crypto is littered with situations where something "good" (such as advantages of avalanche) can't be proven, but something "bad" (such as avalanche not being sufficient) can be easily proven. For example, consider a block cipher where each input bit is XORed into exactly half of the output bits under control of the key. It has perfect avalanche, and yet is trivially breakable with any linearly independent set of known plaintexts.

Feistel structure (2)

- Break input into halves
- use a nasty transformation of one half to modify the other half
- swap halves
- repeat until headache goes away



17-May-01 Copyright Greg Rose, 2000-2001 slide 26

Luby-Rackov

- Suppose you have a set of "perfect" random functions *f* (selected by the key)
  - "Random Oracle" model of proof
- Then four rounds of Feistel, using 4x*f* for the miracle, has been proven to have a certain minimum strength
  - strength is only 1/4 of blocksize, but hey, it's provable!
  - Attack assumes a dictionary function outputs and the birthday paradox
- This is called the Luby-Rackov construction

17-May-01 Copyright Greg Rose, 2000-2001 slide 27

Of course the bad news is that not only are there no known "perfect" random functions, but there is currently no proof that such a thing can even exist in theory. Many people believe that this is either undecidable or just plain false.

However, the concept of "perfect" random functions is nevertheless useful. Many of the proofs we have are under "the random oracle model" where you assume, for the purposes of the proof, that there is a supreme being who can give the same answers predictably all the time, but there is no way to get any information at all about what such an answer might be.

Naor and Reingold

- A relatively recent result shows that the outer rounds of 4-round L-R perform a different function than the inner ones.
- They have different, weaker, requirements, and are more important for mixing than absolute security
  - c.f. design of MARS
  - "whitening" in most new ciphers and IP of DES

17-May-01 Copyright Greg Rose, 2000-2001 slide 28

M. Naor and O. Reingold, *On the Construction of Psuedo-Random Permutations: Luby-Rackoff Revisited"*, Proc. 29th ACM Symposium on Theory of Computing, 1997.

Feistel structure (3)

- Easily reversible, since the function *f* can be recalculated at every stage
- Doesn't require *f* itself to be invertible
- Decryption, assuming *f* uses key material, simply need to feed key material into the encryption function in reverse order!
- Generally a bad idea for the key material to be *un*related between rounds
  - "Key scheduling" is the process of turning a key into the required subkeys

To decrypt you also have to "undo" the last swap of halves, but I'm going to ignore this from here on, since I've just mentioned it...

The last point bears repeating, as it is quite counterintuitive.

Take for example DES. The 56 bit keys are expanded to 16, 48-bit subkeys. For a long time it was assumed that feeding in independent subkeys would be a cheap and easy way to increase the strength of DES -- but in fact it has no good effect, and if the keys are badly chosen actually opens up new attacks (such as the slide attack).

Differential Cryptanalysis

- look at what happens to 2 blocks which are *mostly* the same
  - differences might be a single bit or a single byte
- follow through what happens to the spread of differences
- look for differences where the probability is abnormally high that the configuration will return to that (or some useful) position.
  - probability bias might be quite small, so long as it is detectable
- These are called *characteristics*

Invented (publicly) by Biham and Shamir, who published a book about it. As soon as this was known, Coppersmith went public about the design of the DES S-Boxes, which are optimised to resist this attack, but disclosing the design of the S-Boxes would have told the world about differential cryptanalysis.

When the difference is zero/nonzero in a larger unit, these are called "truncated differentials", pioneered by Lars Knudsen.

What then?

- When you have characteristics, you can *pile them up*. If the cumulative probability is still detectable, you win.
- Get close to the last round, then guess the key bits of the last round(s)
  - divide and conquer attack
  - this is why DES with large key schedule doesn't work

Variations

- Truncated differentials
  - this is where you have larger units, and only know whether the difference is there or not
- Higher order differentials
  - where the difference is some non-linear function of some of the bits
  - not yet practical?
- Impossible differentials
  - look for characteristics with *low* (preferably 0) probability

Linear Cryptanalysis

- Invented by Matsui
  - first *actual* attack against DES
- Approximate non-linear components with linear functions
- Accumulate lots of information
- Try solving the very large set of linear equations
- Check whether the assumed key bits work

Details!

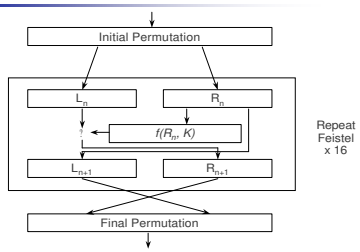
- Goal here is to promote *understanding*
- All these algorithms are defined in detail elsewhere
  - so I won't reproduce S-Boxes, code, bit positions, etc.
  - references given in notes below.
- Order of presentation is (hopefully) to increase understanding, not because of any bias.

DES in more detail

- 64 bit block cipher, 56 bit key
  - Note key is defined as 8 characters with even parity
  - Parity bit is *least significant bit*
  - feeding in ASCII characters is a really bad mistake
- 16 Feistel rounds
  - 8 rounds by the AES terminology
- Initial and Final (inverse) bit permutations
  - slow down software implementations
  - otherwise thought not to be useful

In fact, a correct DES implementation is supposed to check the parity and return an error when the incorrect key is presented. At least one software package did this, but failed to actually do the key schedule in this case… which meant that 255/265 random 64-bit keys were actually equivalent to the all-zeros key. This error wasn't caught for a few years, until someone tried to make the implementation inter-operate with another one, since it of course passed all the test vectors perfectly. This is further proof that it is rarely the algorithm itself that is the real problem.

DES structure



Note that the last "half-swap" is undone. This, plus the fact that the initial and final permutations are inverses of each other, allows the encryption and decryption operations to be identical, except the order of the round keys is reversed.

DES' Key Schedule

- … is basically uninteresting
- Each round uses a 48-bit subkey
- the 48 bits are a different subset of the 56 bits, scrambled
- it is important to the security of DES (and other block ciphers) that the key bits are *dependent* on each other.

17-May-01 Copyright Greg Rose, 2000-2001 slide 37

When they (re-)discovered Differential Cryptanalysis, Biham and Shamir were surprised to discover that DES with independent sub-keys was no stronger.

DES' *f* function

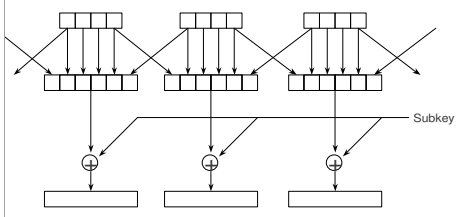
- four stages
  - expand 32 bit half to 8 x 6-bit blocks (48 bits total)
  - XOR with round subkey
  - pass each 6-bit block through an S-box
    - reduces back to 32 bits
  - permute the bits to promote avalanche

17-May-01 Copyright Greg Rose, 2000-2001 slide 38

A paper written by Don Coppersmith just after the re-discovery of differential cryptanalysis by Biham and Shamir details exactly how the expansion and S-boxes were designed to strengthen DES. These details are beyond this tutorial.

Note that, because the S-box substitution loses information, this part of DES is truly irreversible.

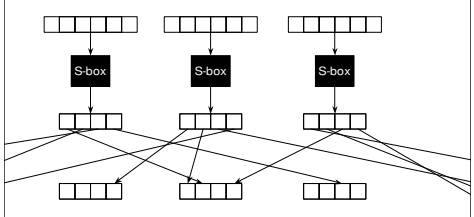
Expansion + Key XOR



17-May-01 Copyright Greg Rose, 2000-2001 slide 39

The "crossover" bits wrap around the ends of the construction, that is, the leftmost bit (of the 32) is copied to the rightmost (of the 48).

S-Box + Permutation XOR



17-May-01 Copyright Greg Rose, 2000-2001 slide 40

There are 8 different S-Boxes, used in parallel (so the three shown above are three different transformations). These are the non-linear component of DES; all the other operations can be treated as systems of linear equations and can be solved directly.

To reiterate, because only four bits come out but 6 bits go in, for each output there are 4 possible inputs. Therefore, the S-Boxes are not reversible, and therefore, the F function as a whole is not reversible. But the Feistel *structure* is still reversible.

Note that the arrows above are just by way of example, they are not accurate.

A paper by Coppersmith gives the actual design criteria for the S-Boxes. ("*The Data Encryption Standard (DES) and its Strength against Attacks*", IBM J. R&D, May 1994.)

Because the design of the S-Boxes was kept secret initially, there were rumours that DES had a trapdoor. (It did -- 56 bit keys!) Revealing the design of the S-Boxes would have told the world about Differential Cryptanalysis, though.

Triple DES

- Run DES three times
  - can use two or three keys (112 or 168 bits of key)
  - Encrypt with K_1, decrypt with K_2, encrypt with K_3
  - For 2-key 3DES, set $K_3 == K_1$
  - For interoperation with (single) DES, set all keys equal
    - that's why the middle operation is decryption
- Use 3DES like a "black box"
  - don't try internal chaining mode
  - *interleaved* modes to speed pipelined operation

17-May-01 Copyright Greg Rose, 2000-2001 slide 41

3DES is a proposed FIPS 46-3:

http://csrc.nist.gov/cryptval/des/fr990115.htm

Based on American National Standards Institute X9.52 standard

Virtually everything you can do with the "innards" of 3DES is broken, so use it as if it is a single algorithm. (See Biham's paper in in Journal of Cryptography.) The exception to this are the "interleaved" modes which allow faster operation through pipelining.

In the interleaved modes, you need 3 initialisation vectors and 3 times as much hardware for this to be useful.

I think 3DES is overrated. It is slow for the results. I'd use one of the AES finalists, preferably Rijndael.

Advanced Encryption Standard

- Block cipher usable for all specified modes, as well as generating Message Authentication Codes (MACs)
- 128 bit blocksize
  - 64 bits too small in a gigabit world
- 128, 192 and 256 bit keys
  - many support even more different sizes
- 15 candidates, winnowed down to 5:
  - MARS, RC6, Rijndael, Serpent, Twofish
- Rijndael won.

17-May-01 Copyright Greg Rose, 2000-2001 slide 42

Reminder: all the detail is at http://aes.nist.gov so I am not going to refer to individual papers.

The 10 rejected candidates were:

Frog (broken)

Magenta (broken at the presentation)

Loki (bad sbox)

Hasty Pudding (no-one could understand it ? , keying weaknesses)

Crypton (problems)

DFC (problems)

SAFER-+ (slow)

CAST-256

DEAL (problems)

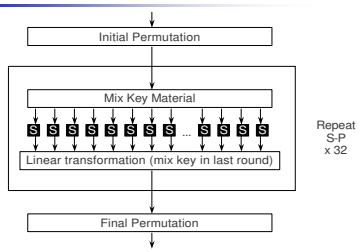
E2

Serpent

- Ross Anderson, Eli Biham, Lars Knudsen
- Arguably the most secure candidate, also generally slower, but very good in hardware
- Implementation trick: uses bitslicing
- Very simple structure, gets security from number of rounds (32).
- Substitution-Permutation Network

17-May-01 Copyright Greg Rose, 2000-2001 slide 43

Serpent structure



17-May-01 Copyright Greg Rose, 2000-2001 slide 44

The initial and final permutations are there only to line up the words correctly for bitslicing… they are actually no-ops in a bitslice implementation.

As in Feistel constructs, where the last swap is not done, the last round is special… it mixes more key material instead of doing the linear transformation (which would be a waste of time).

Bitslicing

- Invented by Biham to speed DES keysearch
- basically, simulate hardware gates with bitwise operations
- but do 32 (or 64, according to wordsize) in parallel
- In Serpent, the block is split into 32 4-bit chunks, which can make use of the bitslicing technique.
- Algorithm is described in more traditional terms.

17-May-01 Copyright Greg Rose, 2000-2001 slide 45

Fast Software Encryption 4 (FSE4), held in Haifa, Israel in January 1997, Eli Biham , "*A Fast New DES Implementation in Software*".

Serpent keyschedule

- Pad shorter keys to 256 bits
- Use to initialise w_{-8} .. w_{-1}
- Use a *linear recurrence relation* to derive 132 more words w_0 .. w_{131}
- Put 4 words at a time through the S-boxes, just like the cipher itself

17-May-01 Copyright Greg Rose, 2000-2001 slide 46

We'll talk more about linear recurrence relations later. This is a recurring theme, other ciphers use similar keyscheduling constructions.

Serpent S-boxes

- There are 8 S-boxes
- Each round uses 32 copies of same S-box
- Round *i* uses S-box $S_{i \bmod 8}$
- S-boxes are derived from DES S-boxes so no-one thinks they're cooked
- Each S-box is 4-bits to 4-bits, permutation
- Inverse S-boxes used for decryption

17-May-01 Copyright Greg Rose, 2000-2001 slide 47

Serpent linear mixing

- 32 bit words $X_{0..3}$ are mixed
  - Rotate X_0 and X_2 (different amounts)
  - Mix X_0 and X_2 into X_1 and X_3 (different ways)
  - Rotate X_1 and X_3 (different amounts)
  - Mix X_1 and X_3 into X_0 and X_2 (different ways)
  - Rotate X_0 and X_2 (different amounts)
- Very easy to parallelise, efficient on superscalar machines like Pentium Pro

17-May-01 Copyright Greg Rose, 2000-2001 slide 48

RC5

- Incredibly elegant cipher
- Split input into halves *a* and *b*, then:
  - a ^= b
  - a = (a <<< b) + *key++
  - swap halves and repeat
- The only nonlinear function is the rotation operation

17-May-01 Copyright Greg Rose, 2000-2001 slide 49

That last bullet requires clarification. "linear" is only meaningful when you define a field or ring or group (see later) for the operations to be defined over. Addition as used in the second operation above is linear, over the ring of integers mod 2^32. XOR as used in the first operation is linear because it is the addition operation over the field GF(2^{32}). However, neither operation is linear in the *other one's* domain. In practice, though, they are both nearly linear at the bit level, and so are considered "sort of linear" when combined as above.

The rotation operation, on the other hand, is very definitely not linear, because trying to write an equation relating any single bit of output to the 32+5 input bits gets very complicated very quickly.

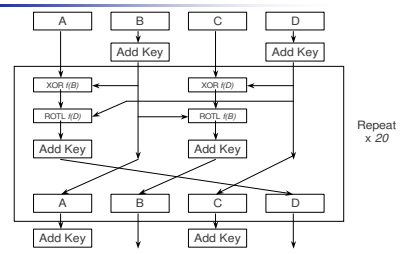
RC6?

- Ron Rivest, Matt Robshaw, R. Sidney, Lisa Yin
- Accepts variable length keys up to 255 bytes
- Very efficient if 32 bit multiply instruction is available; quite slow otherwise
- No S-boxes; nonlinearity comes from data-dependent rotation
- Based on RC5; key schedule is the same.
- Feistel-like; in each round, half of the block is unchanged but determines changes in other half.

17-May-01 Copyright Greg Rose, 2000-2001 slide 50

Trademark of RSA Data Security Inc.

RC6 Structure



17-May-01 Copyright Greg Rose, 2000-2001 slide 51

Whitening

- The process of adding including some key material at the beginning and end of the encryption process is called "whitening".
- This is a cheap way to make it difficult to mount various attacks
  - for example, differential cryptanalysis requires particular bit differences in chosen plaintexts, but this means that you don't actually know what went in.
- RC6 and Twofish use it. MARS… well.
- c.f. the Naor and Reingold result before.

17-May-01 Copyright Greg Rose, 2000-2001 slide 52

RC6 *f( )* function

- Only identified problem with RC5 was that only the low order 5 bits affected the rotation
- This function ensures that all bits of input affect rotation
- *f(x) = (x * (2x + 1)) <<< 5*
- 5 is $\log_2(wordsize)$
- Multiplication result is modulo $2^{wordsize}$.

17-May-01 Copyright Greg Rose, 2000-2001 slide 53

Note that for the multiplication not to lose information, one of the operands must be odd. This is forced by using "2x+1".

Twofish

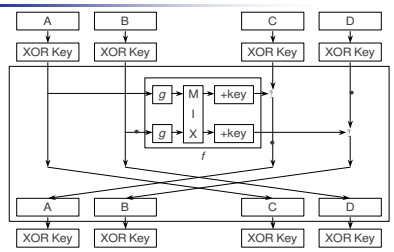
- Schneier, Kelsey, Whiting, Wagner, Hall, Ferguson
- Not really any relation to Blowfish
- Much of the design relies on coding (communication) theory
  - Minimum Distance Separable matrix multiplies
  - Psuedo-Hadamard Transform
- Implementation tradeoffs for key agility or raw speed
- Close to standard Feistel

17-May-01 Copyright Greg Rose, 2000-2001 slide 55

The coding theory stuff is *way* beyond the scope of this tutorial. A good reference is Proakis' "*Digital Communication*"

Twofish structure

16 rounds



17-May-01 Copyright Greg Rose, 2000-2001 slide 57

There are a couple of bitwise rotations left off this slide, because I couldn't fit them on. Basically, B is rotated left 8 bits before going into *g*, and D is rotated left 1 bit before the XOR, and C is rotated right 1 bit after the XOR. The positions are shown as blobs above. These amounts never synchronise.

Psuedo-Hadamard Transform

- *(x, y) = (x+y, x+2y)*
- Automatically achieves good mixing of two quantities
- Easily implemented (and reversed):
  - *x += y*
  - *y += x*
- First used for cryptography (I think) by Massey in SAFER

17-May-01 Copyright Greg Rose, 2000-2001 slide 59

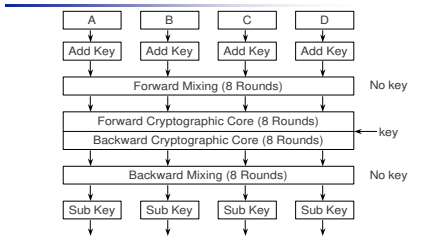
MARS Structure

- MARS uses Naor-Reingold theory
  - outer layers do mixing and avalanche only, no key
  - inner layers are “cryptographic core”
- Structure has “forward” and “reverse” ops
  - ensures that chosen-ciphertext attacks are as hard as chosen-plaintext.
- The combination of different round types and non-linear operations “future-proofs” the cipher.

17-May-01 Copyright Greg Rose, 2000-2001 slide 62

By “future-proofing” they mean that MARS can credibly claim to be resistant to certain kinds of attack that are not (publicly) known yet. Combining the two different methods in the light of the Naor and Reingold construction has some thoretical reasons why even getting to the cryptographic core will be difficult.

MARS Diagram



17-May-01 Copyright Greg Rose, 2000-2001 slide 63

The forward/backward terminology is a bit confusing to me, since the words continue to move in the same direction (see next couple of slides), but the operations are applied differently.

AES (Rijndael)

- Rijndael won the AES competition
- Most efficient in hardware/software/smartcards
- Easily analysed, nice mathematical structure
- More rounds for longer keys was a smart move
- Very good “key agility”
- No “runners up” were announced; none of the finalists were thought to be insecure.

17-May-01 Copyright Greg Rose, 2000-2001 slide 67

Key agility means that it is very fast to do the key scheduling compared to encrypting a block. Bellovin studied the use of keys in IPSec, and found that on average in a backbone router a key would only be used twice before needing to be changed again. So without key agility, either a lot of time, or a lot of memory, would need to be used. It is also useful in memory-constrained devices like smarcards.

My guess about the others

- RC6 requires fast multiply, expensive hardware
- MARS also needs multiply, and memory for key schedule
- Twofish had a “key separation” doubt
- Serpent too slow because of too many rounds (not their fault) – probably the second choice

17-May-01 Copyright Greg Rose, 2000-2001 slide 68

Rijndael

- Joan Daemen and Vincent Rijmen (Belgium)
- Generally fastest of the candidates
- Uses only XOR and table lookups
  - table lookups implement operations over GF(2^8)
  - some table lookups combine multiple operations
- Based on *Square*
- Supports keys which are a multiple of 32 bits, and block sizes which are multiples of 64 bits.
- More rounds for longer keys (10, 12, 14)

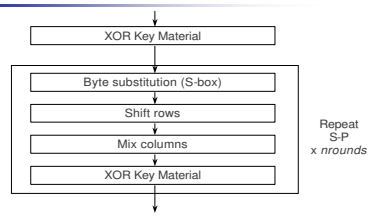
17-May-01 Copyright Greg Rose, 2000-2001 slide 69

"*The block cipher Square*", Daemen, Knudsen and Rijmen, Fast Software Encryption 1997.

We talk more about Galois Fields later.

At the AES3 conference, Rijndael (with extra rounds) was the second choice of all the teams except IBM, who wouldn't say… interesting. It's my favourite.

Rijndael structure



17-May-01 Copyright Greg Rose, 2000-2001 slide 70

As in Feistel constructs, where the last swap is not done, the last round is special… it doesn't bother doing the "mix columns" operation (which would be a waste of time). I don't understand why they bother with the "shift rows" operation either, but they do.

Rijndael keyschedule

- Similar to Serpent's
- Use to initialise w_{-n} .. w_{-1}
- Use a *linear recurrence relation* to derive more words w_0 .. w_{nkey}
- Except that every so often a word is put through the S-box to make it non-linear
- Decryption key schedule takes longer to compute (inverse of MixColumn is not so simple)

17-May-01 Copyright Greg Rose, 2000-2001 slide 71

We'll talk more about linear recurrence relations later. This is a recurring theme, other ciphers use similar keyscheduling constructions.

Rijndael ByteSub (S-box)

- There is only one S-box, 8-bit to 8-bit, called "the ByteSub transformation"
- Applied to each byte in the block equally
- $S(x) = A*(x^{-1}) + b$, where A is an invertible boolean matrix, and x and b are boolean vectors, and the inverse is over GF(2^8)
  - of course implemented as a simple table lookup
  - quite non-linear

17-May-01 Copyright Greg Rose, 2000-2001 slide 72

Rijndael ShiftRow

17-May-01 Copyright Greg Rose, 2000-2001 slide 73

When used on bigger blocks, the shift amounts are different.

Of course this is a simple byte moving operation.

Rijndael MixColumn

- "spreads out" each byte into the entire column
- Treat the bytes as coefficients of a polynomial
- Multiply by another polynomial
  - $3x^3 + x^2 + x + 2$
  - take remainder modulo $x^4 + 1$
  - all bytes are elements of GF(2^8)
  - Alternatively, think of it as a matrix multiply

17-May-01 Copyright Greg Rose, 2000-2001 slide 74

Rijndael magic

- These operations can be combined into simple table lookups and XOR operations
- 4 lookups and 4 xors per column per round
- Can be very effectively parallelised

17-May-01 Copyright Greg Rose, 2000-2001 slide 75

Modes of Operation

- Block ciphers can be used in a number of different *modes*
- Modes have different characteristics:
  - error propagation, resiliency
  - resynchronization
  - character or block resolution
  - efficiency
  - increase in data size
- ECB, CBC, CFB, OFB are defined in FIPS

17-May-01 Copyright Greg Rose, 2000-2001 slide 77

Federal Information Processing Standards (FIPS) are online at:

http://www.itl.nist.gov/fipspubs/index.htm

FIPS 46-2 is the current definition of DES. FIPS 46-3 is supposed to extend DES to Triple-DES, but is not actually an accepted standard yet (April '00). Nevertheless you can get it. FIPS 81 specifies the above modes of operation.

As well as the above, Counter Mode is well accepted (and discussed later). There are also some Interleaved modes defined in ANSI X9.52; these have no more security than the ones above, but allow interleaved operation for triple-DES to increase throughput.

There is supposed to be a FIPS in the works to specify modes of operation for AES, when selected. As far as has been publicly discussed, it will include all of the above.

Block Cipher Modes(1) ECB

- Electronic Code Book: separately encrypt each block
- Gross patterns in input can be recognised
- Same data encrypts same way each time
  - copies of files, same addresses, can be recognised
- Can build up a “codebook”
- Can replace things in the middle
  - both good (database records) and bad (attacks)
- At least it doesn’t need an IV!

17-May-01 Copyright Greg Rose, 2000-2001 slide 78

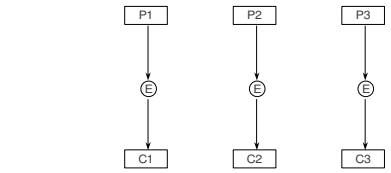
I once caused alarms to go off by (stupidly) pointing out the patterns in some ECB mode encrypted data… it was the blocks of spaces at the end of 80 column card images.

“Oh, look,” sez I, “card images in ECB mode!”…

Man in green suit appears.

“What a nice M16! Could you point it somewhere else? Please?”

ECB Diagram



17-May-01 Copyright Greg Rose, 2000-2001 slide 79

Modes(2) - CBC

- Cipher Block Chaining - XOR plaintext with previous ciphertext block, then encrypt.
- Use an Initialisation Vector (IV) for the first block
- Makes identical inputs look different
- Data corruption self-corrects after two blocks
- Output depends on all previous input
  - can be used as a MAC
- Modifications have some predictable results
- This is mostly what you want to use

17-May-01 Copyright Greg Rose, 2000-2001 slide 80

The Initialisation Vector does not add secrecy… you can send it with the message, or calculate it. However it really should change, preferably randomly.

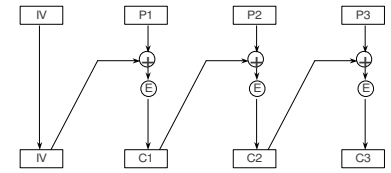
Can’t replace just a small chunk, it will always screw up more than you wanted.

Don’t use the same key for the secrecy and the MAC. There are subtle problems. There is no free lunch.

A one bit change to the ciphertext wrecks the first block, but changes only that bit in the next block.

All generalisations are false, except this one.

CBC Diagram



17-May-01 Copyright Greg Rose, 2000-2001 slide 81

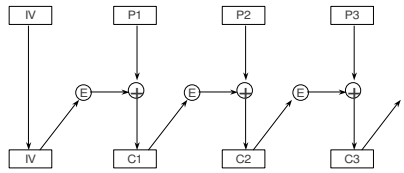
Modes(3) - CFB

- Ciphertext Feed Back; take previous ciphertext, encrypt, then XOR with plaintext
- Don't have to feed back whole blocks. Can be used byte-at-a-time, e.g. telnet.
- Corruptions have unpredictable effects, but still self-correct; sync errors correct too.
- Otherwise much like CBC.
- Less efficient; 8 encryptions to send 8 bytes.

17-May-01 Copyright Greg Rose, 2000-2001 slide 82

To send, say, one encrypted byte at a time, start with the IV, encrypted. Use the least significant byte of the block, XOR with the plaintext, and send that byte. Then shift the block right one byte, put the byte you just sent in the top, and loop.

CFB Diagram



17-May-01 Copyright Greg Rose, 2000-2001 slide 83

It isn't obvious from the diagram, but you don't have to use the entire block of output in each operation. For example, you might use just one byte at a time. In that case, you would shift the IV by one byte, putting C1 into the end, and make that block the input to the encryption. This way, even if a block goes missing entirely, after enough ciphertexts to fill a whole block, you are back in sync.

Modes(4) - OFB and counter

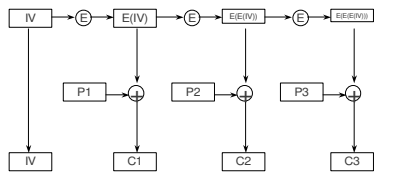
- Output FeedBack - Encrypt the previous output, then XOR with the plaintext to get ciphertext.
- Counter mode - start somewhere, and increment, encrypting at each stage, then XOR with the plaintext.
- These are effectively ways to turn a block cipher into a stream cipher
  - and have all the same problems in practical use

17-May-01 Copyright Greg Rose, 2000-2001 slide 84

Note that decryption of OFB and Counter Modes do *not* require decryption operations, just further encryptions. Therefore it is possible to use non-reversible hash functions in these modes too. No-one is totally comfortable with this construction though.

Rijndael and IDEA (S-P networks both) take longer to set up the key schedule for decryption than encryption, so these modes might be preferred.

OFB Diagram

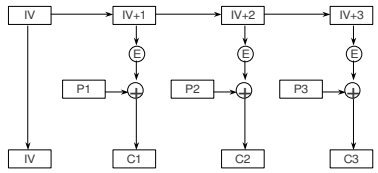


17-May-01 Copyright Greg Rose, 2000-2001 slide 85

When the encryption function is good, you'll get lots of long cycles, and the occasional shorter cycle. A good reference for this kind of behaviour is Menezes, van Oorschot and Vanstone, "Handbook of Applied Cryptography".

With appropriate buffering, you can encrypt byte-at-a-time without the loss of efficiency of CFB Mode. However sync errors are very bad again.

Counter Mode Diagram



17-May-01 Copyright Greg Rose, 2000-2001 slide 86

One of the nice things about counter mode is that the cycle length is guaranteed to be as long as it takes for the counter to wrap.

With appropriate buffering, you can encrypt byte-at-a-time without the loss of efficiency of CFB Mode. However sync errors are very bad again. Counter mode proves to be very useful in applications where external synchronisation is forced, though, for example the STU-III (US Gov't Secure Telephone Unit)

New Modes of Operation

- NIST is hosting a workshop in August
- Proposed modes are available for examination at:
  - http://csrc.ncsl.nist.gov/encryption/modes/
- Some of the modes include one-pass encryption+MAC
  - I think all of these are patented, though.

17-May-01 Copyright Greg Rose, 2000-2001 slide 87

Stream Ciphers

- Generate pseudo-random numbers, XOR with the plaintext to get ciphertext, and vice versa.
- What could be simpler?
- Hard to use correctly:
  - must never reuse a key
  - no protection against modification, no binding
  - effect of modification is predictable
  - some attacks (e.g. precomputation, known plaintext) are easier
- E.g.: A5, RC4, SOBER, WAKE, SEAL, Panama

17-May-01 Copyright Greg Rose, 2000-2001 slide 88

If you reuse a key, the same stream of output is generated. If you take two encrypted samples and XOR them together, the stream cipher cancels out, and you have two chunks of data XORed together. This is called a running key cipher, and is surprisingly easy to break using statistical properties of the plaintexts. (It doesn't matter *how* you combine the stream, there's always (by definition) a way to make it cancel out.)

Because you get nothing for free, you always must use stream ciphers together with other things; random numbers to enhance keys, some way to agree on keys, MACs or hashes to authenticate and bind data, some way to guarantee synchronisation.

A5 is the encryption algorithm used in GSM digital phones. The version which was publicly known was not actually the real thing, which was released in April '99. It is free, but slow in software, and only has about 40 bits of security. A weakened version, A5/2, was revealed in August '99. Biryukov, Shamir and Wagner have laid A5/1 to rest (Fast Software Encryption, 2000).

RC4 was kept a trade secret by RSA Data Security Inc, but leaked somehow; there are a number of "cleanroom" implementations. It is very fast, but its legal status is murky. SOBER is free for non-embedded use, and designed to be fast in software on various size microprocessors. There are less stream ciphers around (in public) than there are block ciphers. Panama is partially broken.

Modes of Stream Ciphers

- Usually, stream is independent of plaintext or ciphertext, and state depends only on previous state.
  - sometimes called OFB mode
- Some stream ciphers update state depending on the plaintext or ciphertext
  - called CFB mode (ciphertext)
  - called autokey cipher (plaintext)
  - autokey systems are usually broken, because there has to be some way to make them fall back to a known state.

17-May-01 Copyright Greg Rose, 2000-2001 slide 89

Note that in the latter case, you are not so much attacking the cipher itself, as the way in which it must be used in practice.

CFB mode stream ciphers must be very strong indeed, because they leave the door wide open for chosen-plaintext attacks.

The intelligence community uses different terminology in this area than is common in the open literature, and it can get quite confusing. What I call *key* they call *cryptovariable*, and what I call *stream* or *keystream* they call *key*.

Modular arithmetic

- I can skip this one, right?
- "clock arithmetic", take remainders of the result when divided by the modulus.
- e.g.
  - 100 + 200 = 44 (mod 256)
  - 1 + 1 = 0 (mod 2) (this is exclusive or, XOR)
- Sometimes *mod* means an environment to work in
- Sometimes it is an operation to perform

RC4?

- Another of Ron Rivest's brilliantly simple designs
- Used in many standards
- Outputs a stream byte-at-a-time
- Variable length key up to 256 bytes
- 258 bytes of state
  - byte permutation (state) table *S*
  - counter *i*, bouncy index *j*
- very fast

More RC4

- Some things to avoid:
  - First outputs correlate with key. Discard 256 bytes.
  - Never use the same key twice (or related keys)
  - After a few gigabytes, you can notice that the same output is duplicated just a little too often
- None of these are problems in practice; SSL is perfectly safe.
- CipherSaber uses it almost exactly wrong.

CipherSaber is a very simple encryption program, sort of a challenge. You are not supposed to find it, rather you are supposed to *implement* it yourself. See http://ciphersaber.gurus.com/ . It's still probably secure, but is not very conservative.

RC4 keying

- Key *k*, key length *l*; all operations modulo 256
- *i = j = 0*
- set *S* to {0, 1, 2, … , 255}
- *repeat 256 times:*
  - *j += S[i] + k[i mod l]*
  - *swap(S[i], S[j])*
  - *increment i*
- *i = j = 0*

Note that since S starts life as a permutation (that is, each value appears in it exactly once), and the swap operation preserves this property, S remains a permutation. There are 256! such permutations possible… a really big number. Even more than the 2048-bit key allows.

Note also that S[0] is set to the first byte of the key, and there's a fairly high chance that it will stay that way. That's one of the reasons why you should discard the first few outputs.

RC4 operation

- To generate the next byte of output
  - *++i*
  - *j += S[i]*
  - *swap(S[i], S[j]);*
  - *output S[S[i]+S[j]]*
- The state array is being continuously shuffled
- *j* jumps around somewhat unpredictably

17-May-01 Copyright Greg Rose, 2000-2001 slide 94

I came pretty close to getting the algorithm right purely from memory (I updated *j* incorrectly).

But now...

- Before we can explore shift register based stream ciphers or public-key algorithms, we need some…

(gulp)

… mathematics

17-May-01 Copyright Greg Rose, 2000-2001 slide 95

Recurrence relations

- Any sequence where n^{th} element is defined by an equation involving previous elements
- Example: Fibonacci numbers:
  - $F_k = F_{k-1} + F_{k-2}$
  - Initial state: $F_0 = F_1 = 1$
  - 1, 1, 2, 3, 5, 8, 13, 21, …
- *Order* of the recurrence relation is number of terms right side "goes back"
- Relation to polynomial equation:
  - $x^2 - x - 1 = 0$

17-May-01 Copyright Greg Rose, 2000-2001 slide 96

This is just the simplest example.

The polynomial form is called *the characteristic polynomial* of the recurrence relation.

General form of a k-th order *linear* recurrence relation:

$$x_n = c_0 x_{n-1} + c_1 x_{n-2} + \ldots + c_{k-1} x_{n-k}$$

Characteristic polynomial:

$$X^n - c_0 x^{n-1} - c_1 x^{n-2} \ldots - c_{k-1}$$

Note that when the coefficients are over a field of characteristic 2, "+" is the same as "-", so people often write the characteristic polynomial with "+" signs and this gets confusing. More later.

The *order* of the recurrence relation is the same as the *degree* (greatest exponent) of the polynomial.

Groups, Rings, and Fields

- For cryptography, we are only interested in *finite* sets
  - Integers modulo another integer
  - polynomials of degree less than an integer
    - with coefficients which are…
- These concepts describe the properties we can make use of, such as being able to add, multiply, divide, exponentiate

17-May-01 Copyright Greg Rose, 2000-2001 slide 97

Of course there are infinite sets too, like the *ring of integers* or the *field of real numbers*. We just generally don't get to use truly infinite things.

Groups

- A (finite) set of elements to operate on
- An operation (call it "+") with the following properties:
  - if *x* and *y* are in the set, so is *x+y* (*closure*)
  - *(x+y)+z == x+(y+z)* (+ is *associative*)
  - There is an element (call it "0") such that *0+x == x+0 == x* (there is an *identity* element)
  - For each x there is an element (call it "-x") such that *x + -x == -x + x == 0* (this is the *inverse*)
- if, as well as above, *x+y == y+x* the group is *commutative* or *Abelian*.

For those with appropriate upbringing, the group properties can be remembered as CAIN (Closure, Associative, Identity, iNverse) and ABEL.

When referring to a group, you get to choose whether you talk about it *additively* as above, or using *multiplicative* notation, where you think of the operator as multiplication, the identity as "1", and the inverse as $^1/_x$ or x^{-1}. This doesn't matter much until you start thinking about rings.

Matrix multiplication is a group most people are familiar with that is *not* abelian.

Rings

- An abelian group G with addition "+".
- A multiplication operator "*" such that:
  - multiplication is associative
  - multiplication *distributes* over addition:
    - *x*(a+b) == x*a + x*b*, and *(a+b)*x == a*x + b*x*
- If multiplication is commutative, so is the ring
- If there is an identity for multiplication, it's a *ring with identity*
- e.g. Integers mod *N*, for any *N*.

Yes, there are rings which don't have an identity element, for example the set of *even* integers… (*1* is not an element of the set, but all the other rules apply). Note that we're talking about the *multiplicative* identity here… there must be an additive identity because G is a group.

Fields

- A field is a set *F*
  - If it's an abelian group for addition
  - and a ring
  - and is an abelian group for multiplication if you ignore 0 (which can't have a multiplicative inverse…)
- The set *F** (*F* leaving out *0*) is itself a group, and is called the *multiplicative group* of the field *F*
  - note: *not* a subgroup -- the operation is different
- e.g. Integers mod *P*, where *P* is prime

sub-thingys

- Given a (group, ring, field), if some subset of its elements forms a (group, ring, field), it is called a (*subgroup, subring, subfield*)
- Note that it is quite possible for a ring to have a subfield (example coming)
- Usually, though, what we care about is when the *thing* has one or more *sub-thingys*.

"*sub-thingys*" is not a very technical term.

Polynomials

- A polynomial of degree *k* is an equation of the form:
  - $a_k x^k + a_{k-1} x^{k-1} + \ldots + a_1 x + a_0$
- the coefficients a_i are usually elements of some field *F*
- the set of polynomials of degree <= *n* forms a field, called the *Galois Field GF(F^n)*
  - addition is termwise within the underlying field
  - multiplication is polynomial multiplication, reduced modulo an *irreducible* polynomial of degree *n*

17-May-01 Copyright Greg Rose, 2000-2001 slide 102

What is *x*? Who cares? Think of it more as a placeholder for sorting out the arithmetic, and not a variable or something to be solved for.

Polynomials generally (when not limited in size, or when the field itself is not finite) form a *ring with identity.*

Evariste Galois died in a duel at the age of about 21… a terrible waste.

Enough of this for now? Good. We'll be back.

An important field: F_2

- The integers modulo 2 form a (small) field.
- Addition modulo 2 is *XOR*
- Multiplication modulo 2 is *AND*
- Integers modulo 2^n ($n > 1$) do *not* form a field
  - even numbers have no multiplicative inverses
- Polynomials of degree *n* over F_2 *do* form a field, *GF(2^n)*.

17-May-01 Copyright Greg Rose, 2000-2001 slide 103

Linear equations

- When the operations are over a field
- … and you have *n* variables
- … and you have *m* equations relating them
- Interesting things happen
- Gaussian Elimination can be used to:
  - reduce the "freedom" of the variables
  - with enough information, down to a solution
  - … or possibly to prove there is no solution

17-May-01 Copyright Greg Rose, 2000-2001 slide 104

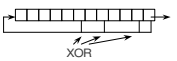
Many aspects of cryptanalysis boil down to describing things so that they act as fields, then gathering enough information to apply Gaussian Elimination. Sean Murphy's result about Twofish mentioned earlier is a sort of example.

If you have *n* independent equations in *n* variables, you have a solution. By *independent* we mean that the equations don't duplicate (or contradict) information you already had. The exact definition of this is too complicated for here.

Karl Friedrich Gauss is who this is named after.

Linear Feedback Shift Registers

- A bunch of bits, obeying a recurrence relation.
- Easily implemented in hardware
- This is called the *Fibonacci* configuration



- A pain in software
- Characteristic polynomial above is
  - $C(x) = x^{12} + x^6 + x^4 + x + 1$

17-May-01 Copyright Greg Rose, 2000-2001 slide 105

The polynomial shown is *primitive* (I know, 'cause Schneier told me so…).

Remember, those "+"s are really "-"s, but that's the same thing in a field of characteristic 2.

LFSRs in software

- There's a software equivalent, called the *Galois configuration*:
- *if (r & (1 << 12))*
  - *r = (r << 1) ^ 0x53;*
- *else*
  - *r <<= 1;*
- Note the difference in shift direction
- Sequence of bits generated is the same, but the *state* of the register at a given point is different

Galois configuration

- If the state of the register is considered as a polynomial
- … the shift left is "multiply by *x*"
- and the XOR is "reduce modulo *C(x)*".

LFSR update

- Updating the LFSR (either style) can be viewed as a matrix multiplication

$$\vec{s}_{n+1} = \vec{s}_n A$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

LFSR sequence length

- 2^n possible states for *n*-bit register
- The all-zeros state stays that way
- If you're lucky, you get a single cycle of the remaining 2^n-1 states; depends on polynomial
  - *C(x)* can be factored, get lots of different cycles
  - *C(x)* irreducible but not primitive, get parallel but disjoint cycles
  - *C(x) primitive*, and *x* is a *generator*, get a maximal length sequence (called an *m-sequence*)
- More about *primitive* and *generator* later

There are many different ways of writing this, pre- or post-multiplication, row or column vector, but the point is it can be done fairly simply, and this equivalence makes all sorts of results easy to figure out.

Note that the first column of the matrix (in this form) is the bits of the recurrence relation.

There are different ways of defining "primitive" and "generator", and in fact the definitions are related in interesting ways. These concepts will come up later, in a context where it is easier to explain them, under Public-key Cryptography.

For the time being, I'll just note that a perfectly good definition of the terms is "if you have a maximal length sequence generated by a shift register, then the characteristic polynomial must have been primitive".

About LFSRs

- All sorts of uses
  - Error correcting codes
  - noise generators in CDMA phones
  - good statistical properties for simulations
  - building block in crypto algorithms
- *but not secure in their own right*
  - if you know polynomial, do Gaussian Elimination
  - if you don't, use Berlekamp-Massey, *2n* bits reveals both the state and the feedback polynomial.

17-May-01 Copyright Greg Rose, 2000-2001 slide 110

Making LFSRs secure

- make the updating "irregular"
- make the output nonlinear
  - requires combining multiple bits from the sequence (called a "combiner with memory")
  - or, equivalently, using a nonlinear function of the current state (called a "(nonlinear) filter generator")
- combine outputs from multiple LFSRs
  - register lengths should be relatively prime
  - output is still linear, just more complicated
  - works well with filter generator

17-May-01 Copyright Greg Rose, 2000-2001 slide 111

The "multiple LFSR" is exactly what you get when you have a larger LFSR with a reducible (factorizable) characteristic polynomial. However the particular state and feedback function of the hypothetical larger LFSR is quite complicated when compared to the individual smaller LFSRs.

A5/1

- Algorithm used in GSM phones (where allowed)
- Developed by GSM companies, kept secret
- Uses irregular clocking and multiple registers
- 64 bit key, bit output, stream cipher
- Was never stronger than about 2^{43}
  - and that attack has been extensively optimized, using time-memory tradeoffs

17-May-01 Copyright Greg Rose, 2000-2001 slide 112

While the algorithm is capable of accepting a 64 bit key, in actual use in GSM it is only ever fed a *54* bit key (10 of the bits are set to zero). No-one knows a way to exploit this other than brute force, and there are other ways to do break it more efficiently than that.

See Biryukov, Shamir and Wagner in Fast Software Encryption, 2000.

A5/1 structure

- 3 registers, (19, 22, 23) bits, maximum length
- output at each stage is XOR of top bits
- "middle" bits of registers control clocking
  - find "majority" of the three clock control bits
  - update only the registers that agree with this
  - at least two registers shift, possibly ($^1/_4$) all three
- Extremely simple in hardware

17-May-01 Copyright Greg Rose, 2000-2001 slide 113

An early version of A5 appeared in public in the early '90s, but there were a number of things about it which were unknown or incorrect. The correct version (which satisfies the test vectors) was reverse engineered in 1999 by the Smart Card Developers Association, see http://www.scard.org .

There is also an intentionally weakened version called A5/2, in which the clocking is controlled by a fourth shift register and a simple nonlinear filter is added, for which a simple divide-and-conquer attack applies (try every possibility for the fourth register, and derive a set of linear equations relating the unknowns of the original three to the observed outputs. With somewhat more than 64 bits of known plaintext, if you can solve the linear equations, you have with high probability have found the initial state).

A5/1 keying

- start with registers zero
- XOR key bits into right bit of registers, and shift (ignoring funny clocking)
  - Note, there are two keys. First load the secret one, then load the "frame number"
- When finished, run it for 100 cycles
- There's a chance one of the registers will end up zero. Surprisingly, this isn't much of a weakness.

17-May-01 Copyright Greg Rose, 2000-2001 slide 114

SOBER

- Greg Rose (yes, me)
- uses LFSRs but with nice size chunks for software
- three versions, for different word lengths
  - t8, t16, t32
- fast keying, small memory, fast generation
- free for non-embedded use

17-May-01 Copyright Greg Rose, 2000-2001 slide 115

Published in Australian Conference on Information Security and Privacy, 1998. The published one had only linear key mixing, which was a bad mistake, and a new kind of brute force attack was developed (Bleichenbacher, Patel, Sundaramen, FSE'99). The current version fixes the former, and optimizes against the latter, and the design paper and source code are online at http://www.home.aone.net.au/qualcomm .

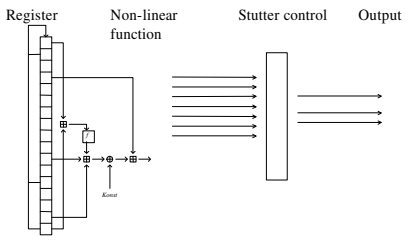
SOBER-t16 and –t32 are being considered for standardisation for the European NESSIE project.

LFSRs over GF(2^8)

- Elements of field are byte-sized binary polynomials
- Addition operation is XOR, ?
- Multiplication is poly-mod multiplication, ?
  - use table lookups
  - only multiply by constants, even better
  - we use $x^8+x^6+x^3+x^2+1$ as the modulus
- Instead of shifting, use pointers to memory
  - sliding window

17-May-01 Copyright Greg Rose, 2000-2001 slide 116

SOBER block diagram



17-May-01 Copyright Greg Rose, 2000-2001 slide 117

The Shift Register

- Generates maximal length sequence
  - period $2^{136} - 1$
- Recurrence:
  - $s_{n+17} = 99 \oplus s_{n+15} \boxplus s_{n+4} \boxplus 206 \oplus s_n$
- Each bit position behaves as output from a 136-bit shift register
  - recurrence relation has 64/136 non-zero coefficients.
- Shift register is "free running"

The Nonlinear Function

- Offsets are carefully chosen
  - taps plus function inputs form "full positive difference set"
- Combine 5 bytes of state and key-dependent value *konst*
  - add two bytes
  - pass through s-box
  - add two more bytes
  - XOR *konst*
  - add last byte

The Stuttering

- Take non-linear value, use it 2 bits at a time to control next four operations

00	No output
01	Output $v \boxplus 0x69$ Cycle register again
10	Cycle register again Output v
11	Output $v \boxplus 0x96$

Keying

- Two stage keying
  - secret key from 4 to 16 bytes (32 to 128 bits)
    - length is significant
    - 3.42e39 distinct keys
    - Save *konst* and register state at this point
  - allows further keying operations (eg. frame)
- Keying process:
  - add key byte to position 15
  - cycle
  - XOR output of nonlinear function to position 4
  - repeat 17 times after key material used up.

Larger word sizes

- There are 16- and 32- bit versions
- Most of the discussion doesn't change
- Field and feedback polynomials are different
- Accept longer keys
- Have more state
- S-boxes are different

Copyright Greg Rose, 2000-2001

See the paper for boring details

Copyright 2000 Greg Rose

S-box

- For t8, the S-box is the f-table from Skipjack
  - permutation
- For the other two:
  - use high order byte to select one entry from a table of 256
  - XOR with low (8, 24) bits of input
  - High byte of table is Skipjack permutation
  - Lower byte(s) are highly non-linear functions developed by Queensland University of Technology

Copyright Greg Rose, 2000-2001

Copyright 2000 Greg Rose

Public-Key Algorithms

- Use keys in matched pairs
  - secret key, sometimes called *d* (for decryption)
  - public key, sometimes called *e* (for encryption)
- Usually very slow (compared to symmetric)
- Often keys are large and/or data expands
- Usually mathematically based
  - rely to some extent on theoretical "hardness" of problems

Copyright Greg Rose, 2000-2001

Copyright 2000 Greg Rose

Inventors

- Ralph Merkle's *Puzzles*
- Whitfield Diffie and Martin Hellman
- Ron Rivest, Adi Shamir, Leonard Adelman
- James Ellis and Clifford Cocks at GCHQ ("Non-secret encryption")
- Hints in NSA documents

Copyright Greg Rose, 2000-2001

Secure communications over insecure channels, Ralph Merkle, CACM April 1978, pages 294-299. Submitted in 1975. No-one understood it at the time.

Diffie, W., Hellman, M. E. (1976). *New Directions in Cryptography*. IEEE Transactions on Information Theory, 22, pp. 644-654.

Rivest, R., Shamir, A., Adleman, L. (1978). *A method for obtaining digital signature and public key cryptosystems,* Communications of the ACM, 21, pp. 120-126

Copyright 2000 Greg Rose

Diffie-Hellman key agreement

- Doesn't actually do encryption or signatures
- Does allow two parties to agree on a shared secret
- … without eavesdroppers being able to figure it out
- However, insecure against active attacks
  - "Man in the Middle"
- relies on the security/hardness of the *Discrete Logarithm Problem*

17-May-01 Copyright Greg Rose, 2000-2001 slide 128

Note that "Man in the Middle" is completely different from "meet in the middle" which we mentioned regarding Triple-DES.

In this, an adversary Mal uses D-H to agree a key with Alice, and another D-H to agree a key with Bob, then faithfully relays messages between them. As far as they can tell, they are talking to each other securely, but Mal can read all the traffic.

Actually, that last point is a lie; it relies on the difficulty of a thing called the *Diffie-Hellman Decision Problem* which is widely believed (but not proven) to be equivalent to the *Discrete Logarithm Problem*.

I want MORE MATHEMATICS!

- Take a prime *P*
- Integers modulo *P*, with the usual definitions, form a finite field
  - can add, and (by taking negative) subtract
  - can multiply, and (by finding inverse) divide
  - can exponentiate by repeated multiplication
  - but can't (easily) undo an exponentiation
- this is the Discrete Logarithm Problem

17-May-01 Copyright Greg Rose, 2000-2001 slide 129

Actually, it's the *Modular Discrete Logarithm Problem* and it's easier to solve than the *General Discrete Logarithm Problem* which we'll hear more about later. This is because as well as multiplying, you can add. In elliptic curves, you only have one operation, and you don't get the help.

Fermat's Little Theorem

- Given a prime *P*, and any *x != 0*,
  - $x^P = x \pmod P$
  - alternatively, $x^{p-1} = 1 \pmod P$
  - (Note: x^{p-2} is the multiplicative inverse of *x*)
- Not to be confused with Fermat's Last Theorem
  - $a^n + b^n = c^n$ has no solutions when $n > 2$.
- A special case of *Lagrange's theorem*

17-May-01 Copyright Greg Rose, 2000-2001 slide 130

Pierre de Fermat also died young, with the statement "I have a truly elegant proof of this which this margin is too small to contain" in the margin of one of his books. It was only recently proven.

It's more efficient to calculate the multiplicative inverse using the extended Euclidean algorithm (later…)

Generators and Order

- The *order* of a group (field) is the number of elements in it
- The order of a group element, is the number of times the operation is applied to it before you get back to the identity
- A *generator* is an element which has the same order as the group
- Remember, mod *P* we are talking about the multiplicative group
- If there is a generator, the group is *cyclic*

17-May-01 Copyright Greg Rose, 2000-2001 slide 131

E.g., multiplication mod 7. The group has 6 elements, 1..6. (Remember, 0 is not part of the multiplicative group of the field.)

1 has order 1.

2 has order 3, $2^1 = 2$, $2^2 = 4$, $2^3 = 8 \pmod 7 = 1$.

3 has order 6: 3^k, k = 1..6, = 3, 2, 6, 4, 5, 1, therefore 3 is a generator.

More about generators

- The order of any element must divide the order of the group
- There are plenty of generators (if any exist)
  - so choose an element at random
  - check if it is a generator...
  - by checking Fermat's little theorem and …
    - this can actually be used as a primality test for *P*
  - check that $x^e != 1$ for $e = (P-1)/q$, for each q which is a prime factor of *(P-1)*.
- Need to know the factors of *P-1*...

Discrete Logarithm Problem

- Given *{1, x, x^2, … x^e, … x^{p-2}, 1, x, …}*
- It's easy to start anywhere and move forward a known number of places
- Given *{1, x, … a, … b, … }*
- it's hard to figure out how far apart *a* and *b* are.

Diffie-Hellman

- Parties agree on prime *P* and generator *g*.
- Parties Alice and Bob choose random *a, b*
- Calculate $A = g^a \pmod P$ and $B = g^b \pmod P$
- Send *A, B* to each other
  - A^b is calculated by Bob
  - B^a is calculated by Alice
  - Both results are $g^{(ab)}$
  - use hash of this as a shared secret key
- No known way to get to the shared secret without calculating a discrete logarithm

Things to watch in D-H

- Don't use g^{ab} directly… hash it first
- *P* is usually chosen so that either
  - *(P-1)/2* is prime (call it *q*)
  - *(P-1)* has a prime factor *q* of roughly twice as many bits as the equivalent symmetric cipher key (160 bits for 1024 bit *P*)
- Look for results that have small order (this is called the *small subgroup attack*); either
  - check that answer isn't +/- 1
  - check that answer has order *q*

I can't remember *why* you are supposed to hash first. There's a good reason.

To avoid the appearance of mounting your own small subgroup attack, you shouldn't choose a totally random value, rather you want one which is of order *q*. This is easy: choose an element at random and raise it to the power *(p-1)/q*, and try again if the answer is 1. The resulting element will have order *q*. If you don't do this, the guy at the other end can figure out up to 1 bit of information about your random number. It sounds trivial...

Discrete Log encryption and digital signatures

- Adaptations of Diffie-Hellman key exchange
- Encryption by Taher ElGamal
- Digital Signatures by ElGamal and Schnorr (DSA)

A Public-Key Cryptosystem and a Signature Scheme based on Discrete Logarithms", T. ElGamal, Crypto '84.

Efficient Signature Generation for Smart Cards, C. Schnorr, Crypto '89

FIPS 186, NIST.

ElGamal Encryption

- PGP calls this "Diffie-Hellman"
- Choose (or agree on) prime P and generator g.
- Choose x which is the secret half of the key
- $y = g^x$ is the public key
- To encrypt M (the message as an integer < P)
  - choose random k
  - calculate $a = g^k$, $b = y^k M \pmod P$; ciphertext is (a,b)
- To decrypt, $M = b.(a^x)^{-1}$
  - D-H key exchange, multiply M by agreed key

Essentially, one party chooses a fixed value to use for Diffie-Hellman key exchange, and publishes the derived value as their public key. The other party does their half of the key exchange (calculating a) and sends that along with the message already encrypted using it (b). The way the message is encrypted is by multiplying it (mod P) with the agreed key, but using it as a key for 3-DES or something would also work.

The recipient simply calculates the shared key (a^x) and undoes the encryption (in the definition of ElGamal, by division).

It's easy to say in retrospect, but I'm surprised that it took 3 years (during which RSA was invented) for this to be figured out.

Digital Signature Algorithm

- Really ElGamal signature, with Schnorr's optimization to make signatures smaller
- There's a defined procedure for choosing the parameters, based on using DES to generate pseudo-random numbers; not relevant.
- First choose 160-bit prime q, then find a large (1024 bit or longer) prime P such that $q \mid (P-1)$
- Choose g to be a generator of the order-q subgroup
- Choose $x <$ q; Public key is $(P, q, g, y = g^x)$

Remember, this is easy: choose an element at random and raise it to the power $(p-1)/q$, and try again if the answer is 1. The resulting element will have order q.

DSA Signature

- Choose random $k < q$
- Hash M to get h
- Calculate $r = (g^k \bmod P) \bmod q$
- Calculate $s = (xr + h)/k \bmod q$
- Signature is (r, s); 320 bits total.
- Schnorr's trick is to do everything mod q, reducing the size (and work) of the signature
- r and k^{-1} don't depend on M; precalculate
- ElGamal and Schnorr multiply not add.

Note that the real work in signature calculation is the exponentiation to form r, so a slow smartcard could easily spend a few minutes doing this in preparation for the next signature, then just do a couple of multiplications when the value to be signed was presented.

Schnorr thinks his patent applies; NIST disagrees because they add instead of multiplying.

Another reason for the scheme is that it makes it harder for DSA keys to be used for encryption, so this was for export control (but it isn't impossible, just harder, so as usual this was a waste of time).

DSA verification

- Basically, calculate the same thing two ways, and check that they agree.
  - $a = h/s \pmod q$
  - $b = r/s \pmod q$
  - $v = ((g^a y^b) \bmod P) \bmod q$
- v is just another way to calculate s, assuming that h hasn't changed
- if $v == s$ the signature is verified

17-May-01 Copyright Greg Rose, 2000-2001 slide 140

It takes a bit of verification to check this. Start by completely ignoring q, and just do everything mod P. This is left as an exercise, because looking at incomprehensible equations does not aid learning, and it was too hard to format it correctly.

RSA (Rivest, Shamir, Adelman)

- Invented in 1977
- Decryption and digital signature generation are the same operation
- as are encryption and signature verification
- Relies on the difficulty of factorizing the composite modulus
  - well, that's not proven
- There are intimate links between factorizing and modular discrete logarithms; currently best algorithms take same time.

17-May-01 Copyright Greg Rose, 2000-2001 slide 141

Rivest, R., Shamir, A., Adleman, L. (1978). *A method for obtaining digital signature and public key cryptosystems,* Communications of the ACM, 21, pp. 120-126

If you can factorize the modulus, you can certainly calculate the decryption key. However there's no proof that there isn't some *other* way to do that, or to recover messages. Mathematicians aren't quite so sure about this one.

More … gondolas

- RSA operations are done modulo $N = pq$, where p and q are primes
- Integers mod N form a *ring with identity* but not a field -- some elements have no inverse
- For elements which have inverses, you need to know p and q to calculate them
- This is where *the Chinese Remainder Theorem* comes in

17-May-01 Copyright Greg Rose, 2000-2001 slide 142

Chinese Remainder Theorem

- Basically, you can decompose an integer mod N into two integers (mod p, mod q), and convert between them. (*isomorphism*)
- There's a straightforward algorithm for converting ($a \bmod p$, $b \bmod q$) to ($c \bmod N$)
- Note that ($1 \bmod p$, $1 \bmod q$) is ($1 \bmod N$)
- The holder of the secret key can use knowledge of p and q to speed up computation
  - since exponentiation time is cubic in modulus length

17-May-01 Copyright Greg Rose, 2000-2001 slide 143

If the remainders are (a, b) respectively, calculate

$c = (((a-b)/q) \bmod p)*q + b.$

To exponentiate, the normal square-and-multiply algorithm does an average 1.5 multiplies per bit. Each multiplication forms partial results proportional to the square of the number of bits. So the total algorithm is cubic. By splitting the calculation into halves, you do twice as many calculations, but on half-sized numbers, for a total saving approaching a factor of 4. This ignores use of Karatsuba multiplication and the overhead to set up and reconstruct the answer, but it's a good rule of thumb.

Fermat applied to composites

- Given M, decompose to ($a \bmod p$, $b \bmod q$)
- $a^{p-1} \bmod p == 1$, $b^{q-1} \bmod q == 1$
  - by Fermat's Little Theorem
- $1^{anything} == 1 \pmod{anything}$
- so $a^{(p-1)(q-1)} == 1 \pmod p$, and
- $b^{(p-1)(q-1)} == 1 \pmod q$,
- but $(1 \bmod p, 1 \bmod q) == 1 \bmod N$
- therefore $M^{(p-1)(q-1)} == 1 \bmod N$
- $M^{(p-1)(q-1)+1} == M \bmod N$

$(p-1)(q-1)$ happens to be the number of integers less than N which are relatively prime to it, and there's a name for this: Euler's *phi* function.

Actually, the second-last point is too specific. Any multiple of phi(N) will do just as well, and in fact $p-1$ and $q-1$ might have factors in common, so any n which is divisible by both $p-1$ and $q-1$ will enjoy the property that:

$M^{n+1} == M \bmod N$

RSA encryption/decryption

- Choose primes p and q, find $N = pq$
- Choose encryption exponent e; since it's going to be public, it may as well be easy, say $e = 2^j+1$ for some small j (eg. 3, 17, 257, 65537)
- Now find any d so that
  - $ed-1 == (p-1)(q-1)$
  - Can't do this unless you know p and q.
- Ciphertext $C = M^e$
- To decrypt, $C^d == M^{ed} == M^{(p-1)(q-1)+1} == M^1 == M$

Even though I used the fixed form of the equation above, to simplify the explanation, any pair $ed == 1 \bmod (lcm(p-1, q-1))$ (where *lcm* is the Least Common Multiple) will work fine.

RSA exponents must be odd. There is an even-exponent encryption algorithm due to Rabin, but the answers are ambiguous (because there are two square roots for every number).

Some RSA "gotchas"

- Never decrypt/sign something an enemy gave you
- Never decrypt/sign the same thing multiple times
- Both of these can be solved by always adding random padding
  - current preferred method is Optimal Asymmetric Encryption Padding (OAEP) (Bellare & Rogaway)

In OAEP, take a random r and hash it to form h. XOR this with the actual message (usually a symmetric key or message digest). Hash *that* and XOR with the random number to form g. Then h and g together become the input to the RSA encryption. It can be proven that if the hash function is good, there's no way to use this to attack RSA.

Things you don't need a key for.

- Hashing
  - also known as "message digest"
  - taking something large and boiling it down
  - collisions are possible, but unlikely
- Compression
  - remove (some) redundancy, but still can recover the original
- Error correction
- Changing representation, e.g.. MIME
- Steganography, "hiding information".

We're not going to look at any of these besides hashing.

Hash functions aka Message Digests

- Most commonly available hash functions are *iterated hashes*
- In these, take an initial state and a chunk of input, put them through a nasty nonlinear process, to produce a new initial state
- It's possible to use block ciphers, but the outputs tend to be too small
  - because of the birthday paradox, hash outputs need to be twice as big as a corresponding cipher block.

17-May-01 Copyright Greg Rose, 2000-2001 slide 148

More about hashing

- Three important characteristics:
  - given hash, hard to find *any* message with that hash
  - given message, hard to find *another* message with the same hash
  - hard to find *any pair* of messages with the same hash
- Hashes are longer than symmetric keys
  - to get the same level of security, need twice as many bits
  - this is because of the "birthday paradox"

17-May-01 Copyright Greg Rose, 2000-2001 slide 149

These three characteristics are called "preimage resistance" ("not reversible"), "second preimage resistance" and "collision resistance", respectively.

Hash History

- Until about 1980, hash functions were rudimentary and (mostly) insecure.
- Secure ones based on a block cipher with chaining and data as key material
  - quite inefficient
- Digital Signatures needed a good Message Digest algorithm.
- Rivest's algorithms: MD2, MD4, MD5, SHA
- Others: RIPE-MD, Tiger, HAVAL

17-May-01 Copyright Greg Rose, 2000-2001 slide 150

MD2 is byte-oriented and relatively slow; as far as I know, it is only used in Digital Certificates, because it is good for smartcards, and for historical compatibility.

MD4 is much more efficient, but Dobbertin found collisions. Even Rivest thought it was "too close to the edge" and had designed MD5 to be more secure before this result was known.

NIST needed a hash with 160-bit output, so they developed SHA (FIPS 180). Soon after, they changed the algorithm to introduce a 1-bit rotation, and this is called SHA-1 (FIPS 180-1) X509 digital certificate implementations are about the only thing that still requires SHA(-0), but even there it doesn't get used in practice.

Dobbertin and others then found a "partial" collision in MD5, and it was clear that the same approach would have worked on SHA(-0); it is assumed this is the undisclosed reason for the update, which solves the problem.

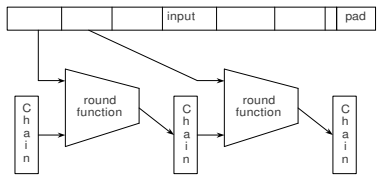
MD5 is "deprecated" -- not broken but shouldn't be used in future developments. There is a proposed "Advanced Hash Standard" with 256, 384 and 512 bit output, to match AES.

Iterated Hash Structure

- Take input message, break into 512-bit chunks, and pad the last one(s).
  - append a "1" bit
  - enough zero bits so that …
  - last thing in last block is 64-bit input length in bits
  - This might require an extra block
- "Compression" or round function takes *chaining value* and one block of input, produces updated *chaining value* as output
  - allows processing as a stream
  - hash output is *chaining value* after last block

17-May-01 Copyright Greg Rose, 2000-2001 slide 151

Hash structure diagram



SHA round function

- Chain value is 5 32-bit words (*A,B,C,D,E*)
- Block converted to 16 32-bit words big-endian
- 16 words extended to 80 words using an LFSR like construction
  - actually, you can use the 16 words as a circular buffer
  - this is where the 1-bit rotation change happened
- run non-linear functions of four variables, a constant, and one input word to modify remaining variable (like unbalanced Feistel)

Function details

- Each operation looks like:
  - *e += (a <<< 5) + f(b, c, d) + constant + data*
  - *b <<<= 20*
  - but rotating the variables around
- functions are run in groups of 20
  - different subfunction and different constant
  - *(b & c) | (~b & d)* (bit multiplexing)
  - *b XOR c XOR d*
  - *(b & c) | (b & d) | (c & d)* (majority function)
  - *b XOR c XOR d*

Chaining details

- At the end of all that, *A,B,C,D,E* are added to the initial chaining value
  - this ensures that you have to “solve equations”
- At the end of all that, chaining value becomes the message digest
  - Note that the digest is *words* not bytes.
  - to compare digests, probably want to convert to bytes in big-endian fashion
  - (I think this is a crock, myself).

SHA-256 (384, 512)

- SHA-256:
  - Similar to SHA-1, but 8 32-bit words
  - Each round updates two of the words based on functions of the others
  - 64 rounds instead of 80 (but more work done)
  - The “rotate 1 bit” fix is now much more complex
- SHA-512
  - Same but with 64 bit variables
- SHA-384 Do SHA-512, truncate result by using 6 64-bit words

Copyright Greg Rose, 2000-2001

Elliptic Curve Cryptography

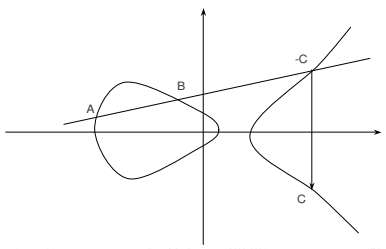
- An *elliptic curve* is the solution set to a general equation:
  - $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$
- For cryptographic purposes over GF(2^n):
  - $y^2 + xy = x^3 + a_2x^2 + a_6$
- With appropriate definitions, the points on the curve form a group with an addition operation

Copyright Greg Rose, 2000-2001

Unlike the polynomials mentioned before, here the *x*s and *y*s do actually get solved for.

A good introductory book is “*Implementing Elliptic Curve Cryptography*”, Michael Rosing, Manning 1998.

Curve over the real numbers



Copyright Greg Rose, 2000-2001

This is a not-terribly-accurate drawing of a curve, approximately:

$y^2 = x^3 - 7x + 5$

The addition rule for two points is, draw a line connecting the two points. Unless the points are directly in line vertically, this line must intersect at exactly one other point (since you are solving a cubic equation). The reflection in the x axis is defined to be the sum of the two points. That is, in the diagram above, C = A + B

The additive inverse of a point is its reflection in the X axis.

The additive identity is the “point at infinity”

To “double” a point, take the tangent at that point.

Curve over finite field

- Much harder to draw…
- The field can be either GF(P) or GF(2^n)
- Nevertheless, the same formulae for intersections and tangents work
- Elliptic curve points form a *group* but not a field (there’s no multiplication operator)

Copyright Greg Rose, 2000-2001

Multiplication and Discrete Logarithm

- Can multiply a scalar and a point, by repeated addition:
  - $7A = A + A + A + A + A + A + A$
- However there's no inverse to this operation
- Solving it is called "the elliptic curve discrete logarithm problem"
  - Why is it called this? This is the hardest thing to understand about elliptic curves.
  - Difficulty seems to be fully exponential, "because" there is only one operation

Elliptic Curve Cryptography

- Equivalents exist for Diffie-Hellman and El Gamal
- Some kinds of curves are weak; finding good curves is hard
- IEEE P1363 has been extended to include some good curves
- Using elliptic curves is not patented
  - however speedup techniques over GF(2^n) may have patent protection

Copyright 2000 Greg Rose

Copyright 2000 Greg Rose