QUALCOMM

# Cryptographic Algorithms Revealed

**Greg Rose**

**ggr@qualcomm.com**

QUALCOMM

# Overview

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

QUALCOMM

# 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!***

QUALCOMM

# 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

QUALCOMM

# 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)

QUALCOMM

# 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

QUALCOMM

# 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?

QUALCOMM

# 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

QUALCOMM

# 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!

QUALCOMM

# 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 (this sentence)!
- Relates to “predictability” and so is relevant to security
  - you have no security if your secret can be guessed
- Good practice to compress first

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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.

QUALCOMM

# 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, …"

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# Block Ciphers

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

QUALCOMM

# 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

QUALCOMM

# 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.

QUALCOMM

# 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...

QUALCOMM

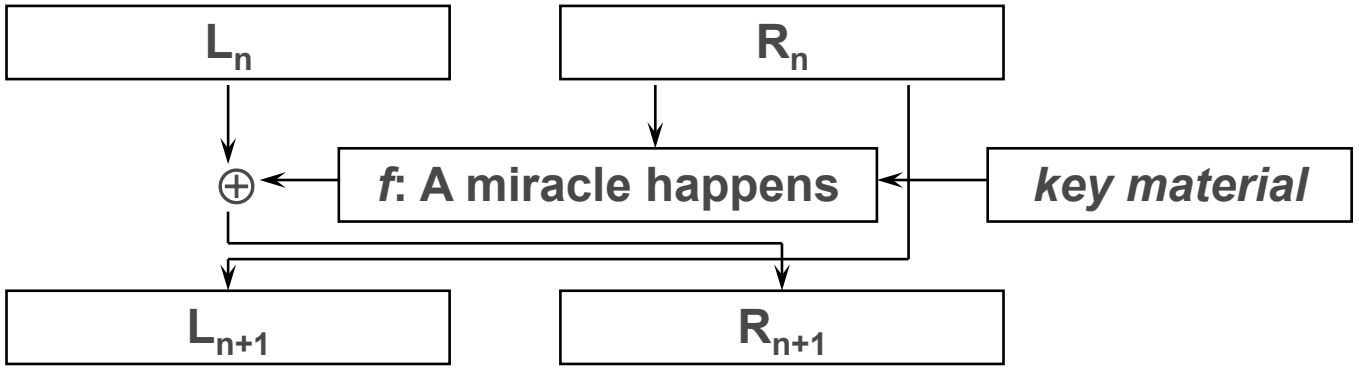
# Feistel structure

- 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.

QUALCOMM

# 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



QUALCOMM

# 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 of function outputs and the birthday paradox
- This is called the Luby-Rackov construction

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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*

QUALCOMM

# 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

QUALCOMM

# 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
- Saturation attacks

QUALCOMM

# 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

QUALCOMM

# Details!

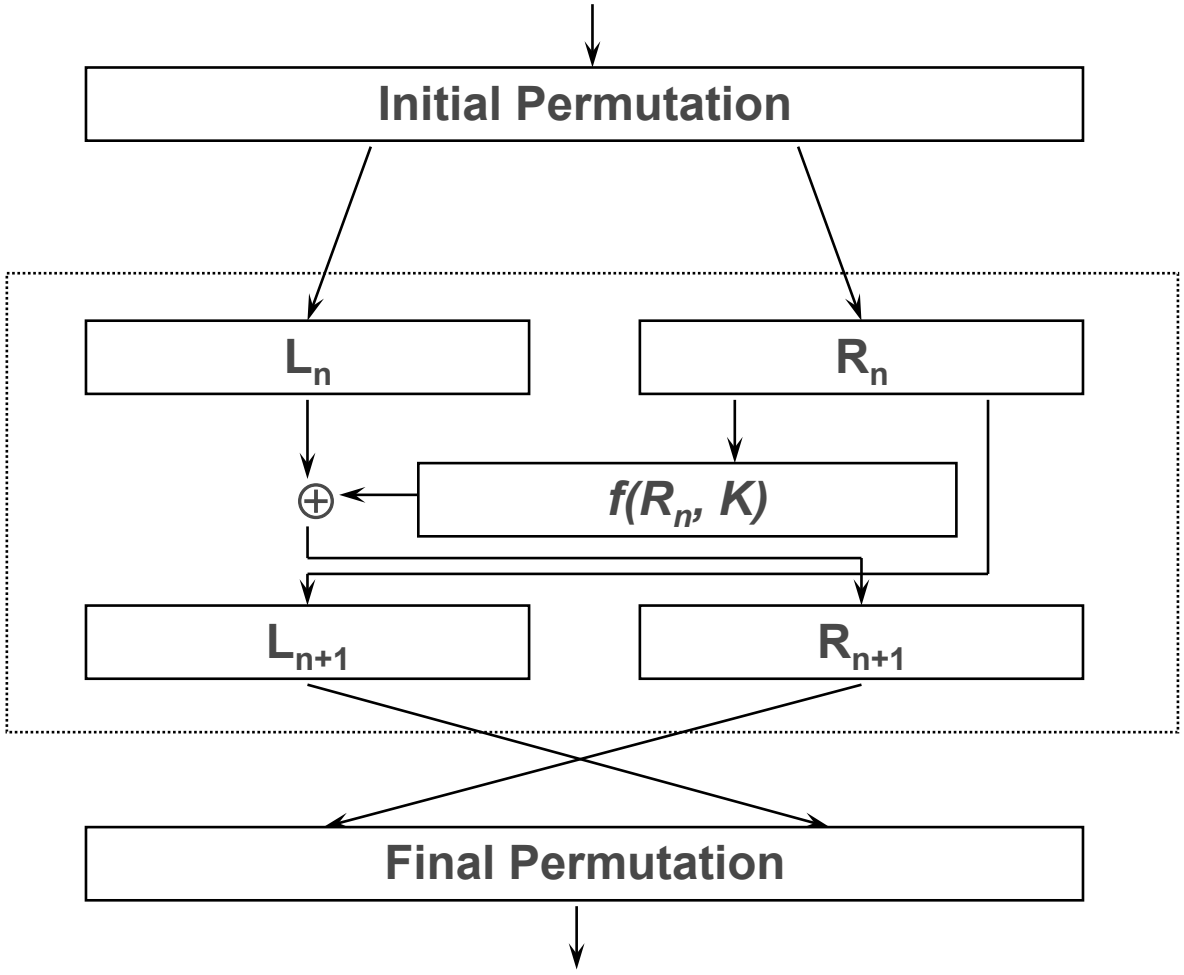
- Goal here is to promote *understanding*
- All these algorithms are defined in detail elsewhere
  - so I won't reproduce S-Boxes, code, bit positions, etc.
  - references given in notes below.
- Order of presentation is (hopefully) to increase understanding, not because of any bias.

QUALCOMM

# 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

# DES structure




Repeat Feistel x 16

QUALCOMM

# 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.

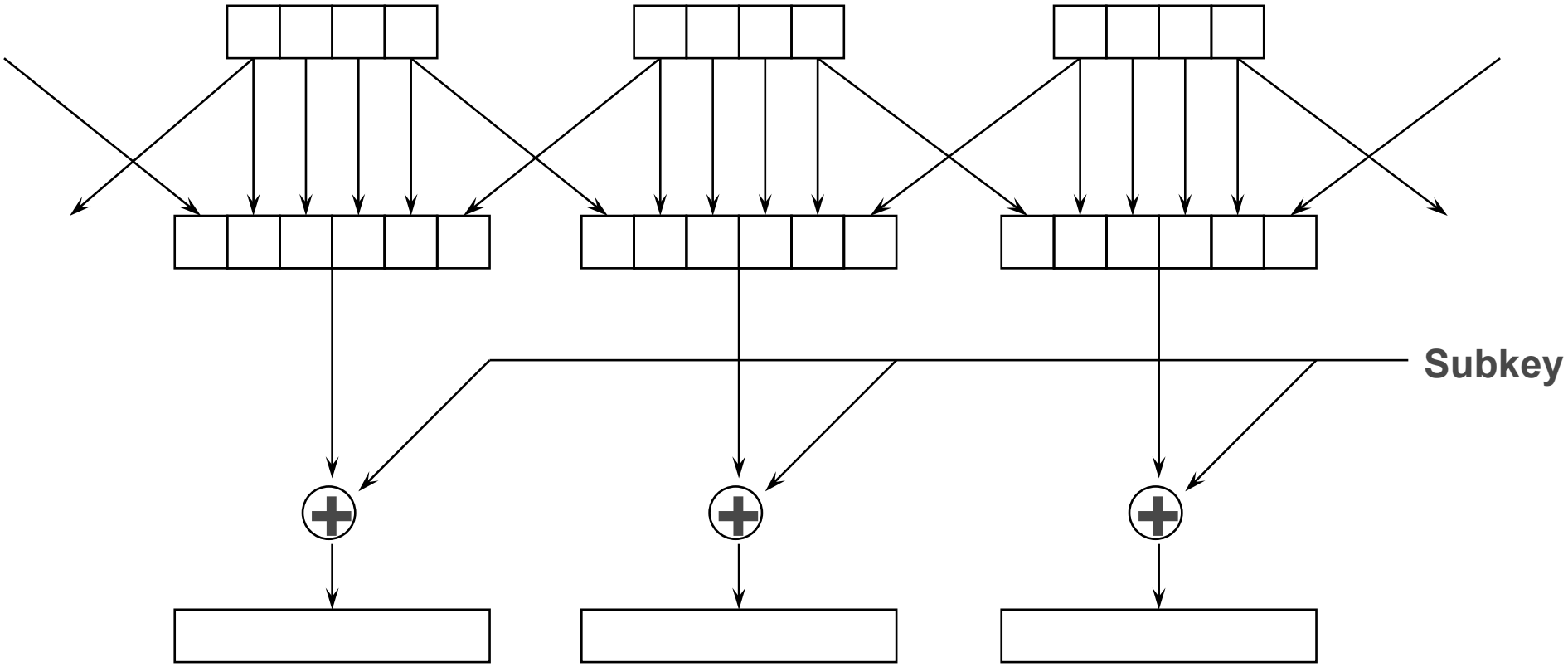
QUALCOMM

# DES' *f* function

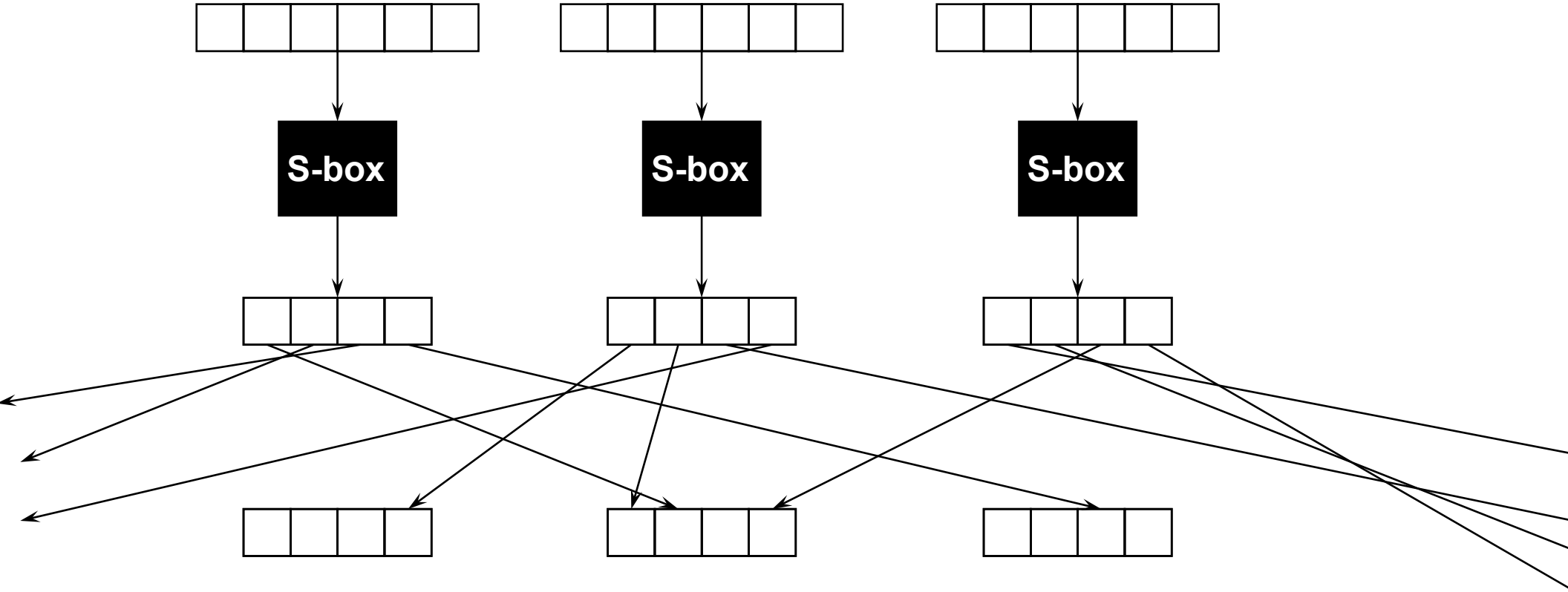
- four stages
  - expand 32 bit half to 8 x 6-bit blocks (48 bits total)
  - XOR with round subkey
  - pass each 6-bit block through an S-box
    - reduces back to 32 bits
  - permute the bits to promote avalanche

QUALCOMM

# Expansion + Key XOR



# S-Box + Permutation XOR



QUALCOMM

# 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

QUALCOMM

# 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.

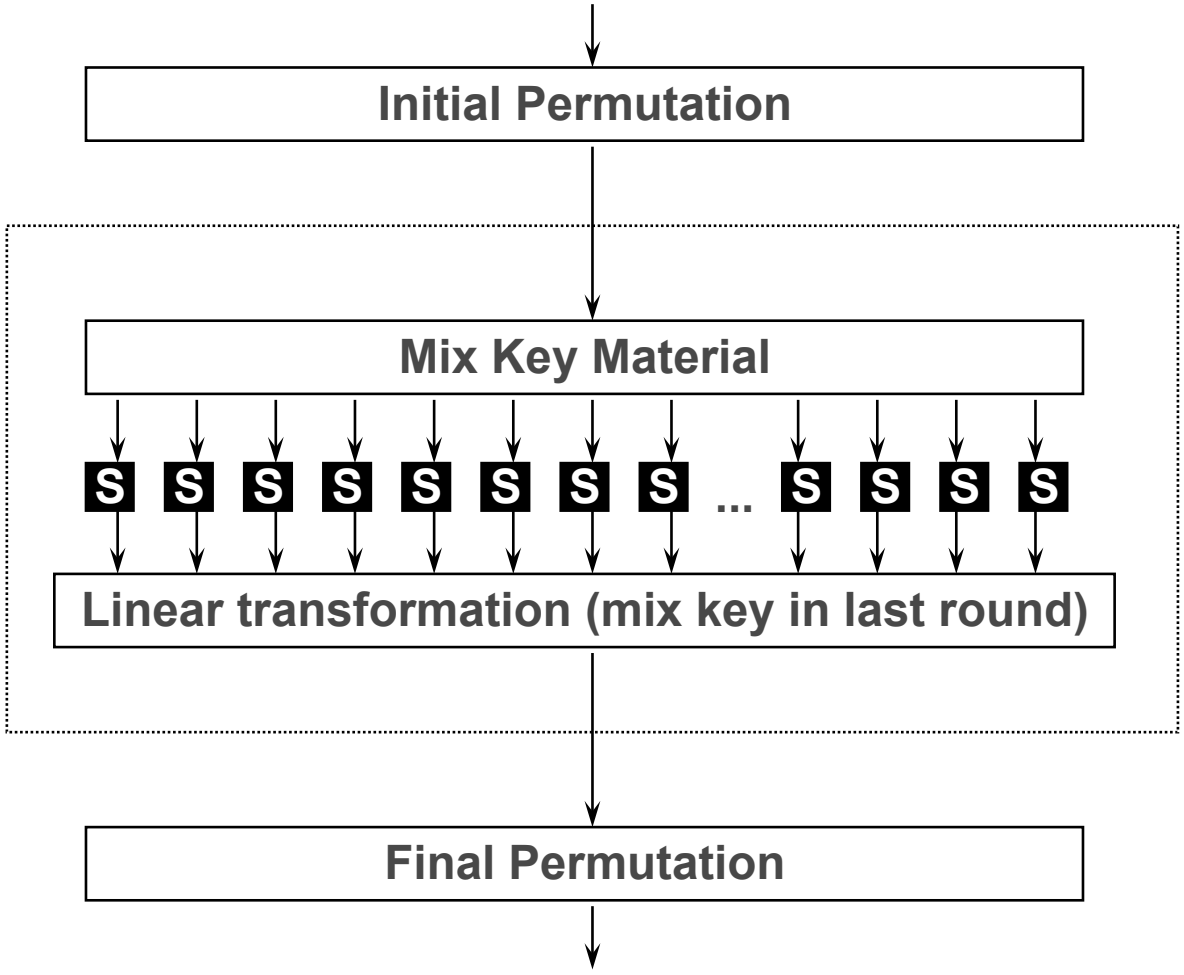
QUALCOMM

# 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

QUALCOMM

# Serpent structure



QUALCOMM

# 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.

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# RC5

- Incredibly elegant cipher
- Split input into halves *a* and *b*, then:
  - a ^= b
  - a = (a <<< b) + *key++
  - swap halves and repeat
- The only explicit nonlinear function is the rotation operation
- Implicit nonlinearity comes from the fact that XOR and add-mod-2^32 are different groups

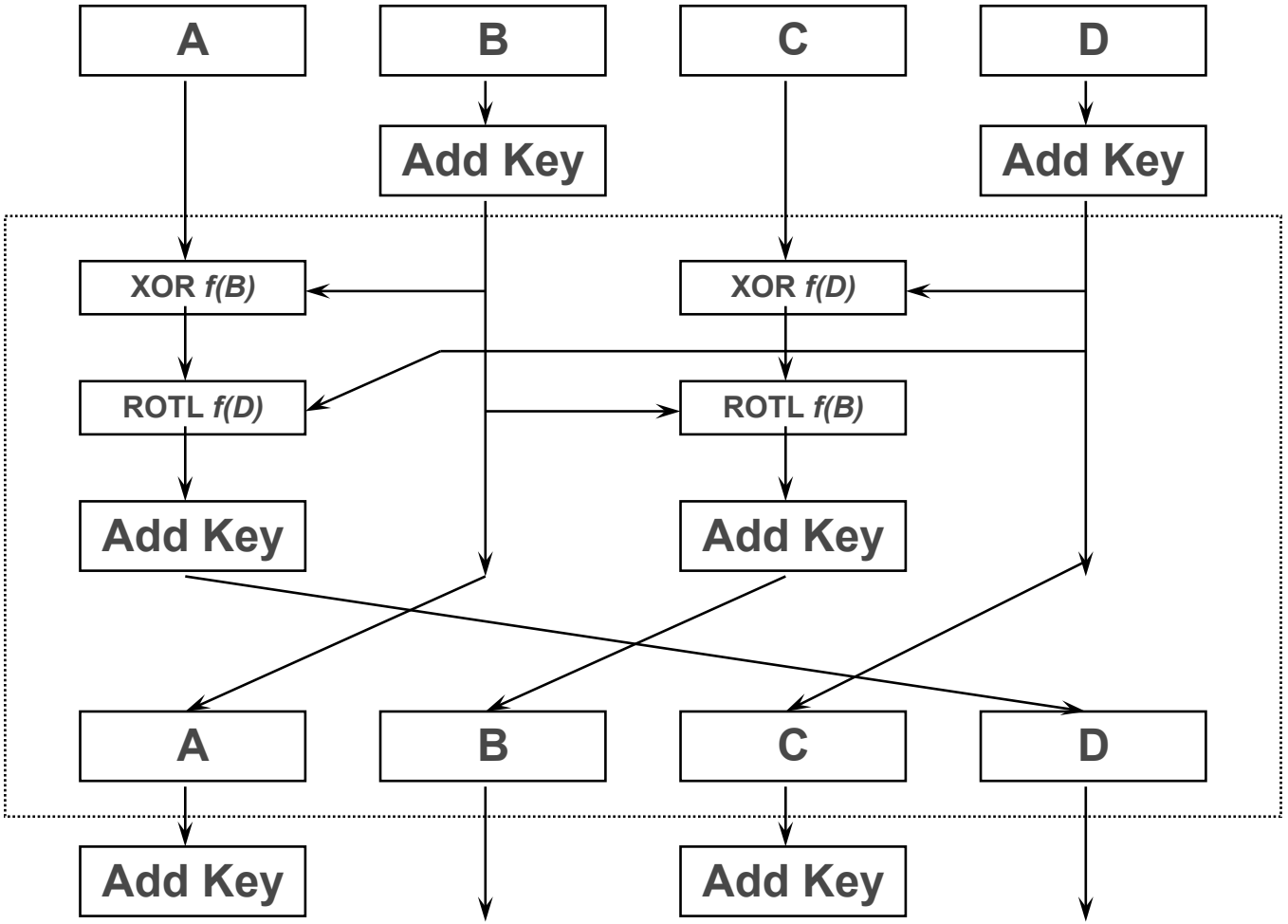
QUALCOMM

# 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.

QUALCOMM

# RC6 Structure



QUALCOMM

# 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.

QUALCOMM

# 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 $2^{\text{wordsize}}$.**

QUALCOMM

# RC6 Key Schedule

- Resembles a hash function
  - Initialise key array with constants
  - make at least 3 passes folding in key materal
- Not (known to be) reversible
  - discovering round keys lets you do encryption and decryption, but doesn't tell you what the input key was.

QUALCOMM

# Twofish

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

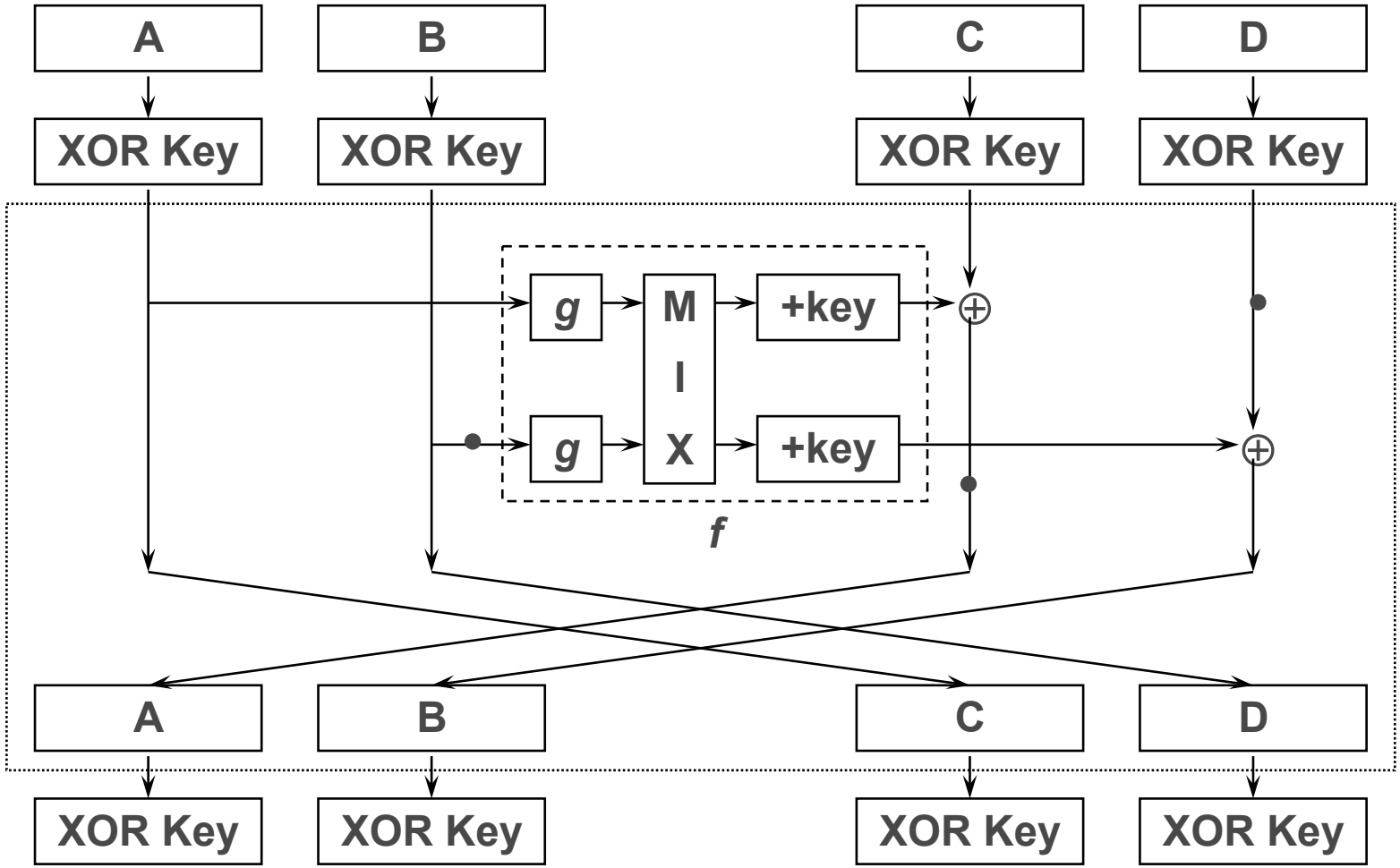
QUALCOMM

# Twofish keying

- 4 8-bit to 8-bit invertible S-boxes
- S-boxes are key dependent and variable
  - calculate on the fly for fast key setup/small memory
  - precalculate (1K memory) for faster encryption
- 64 bits of key entropy goes into S-boxes
  - linearity considerations might allow divide-and-conquer attacks
  - Murphy thinks this is a problem; NIST might too
- Key schedule is "complicated"

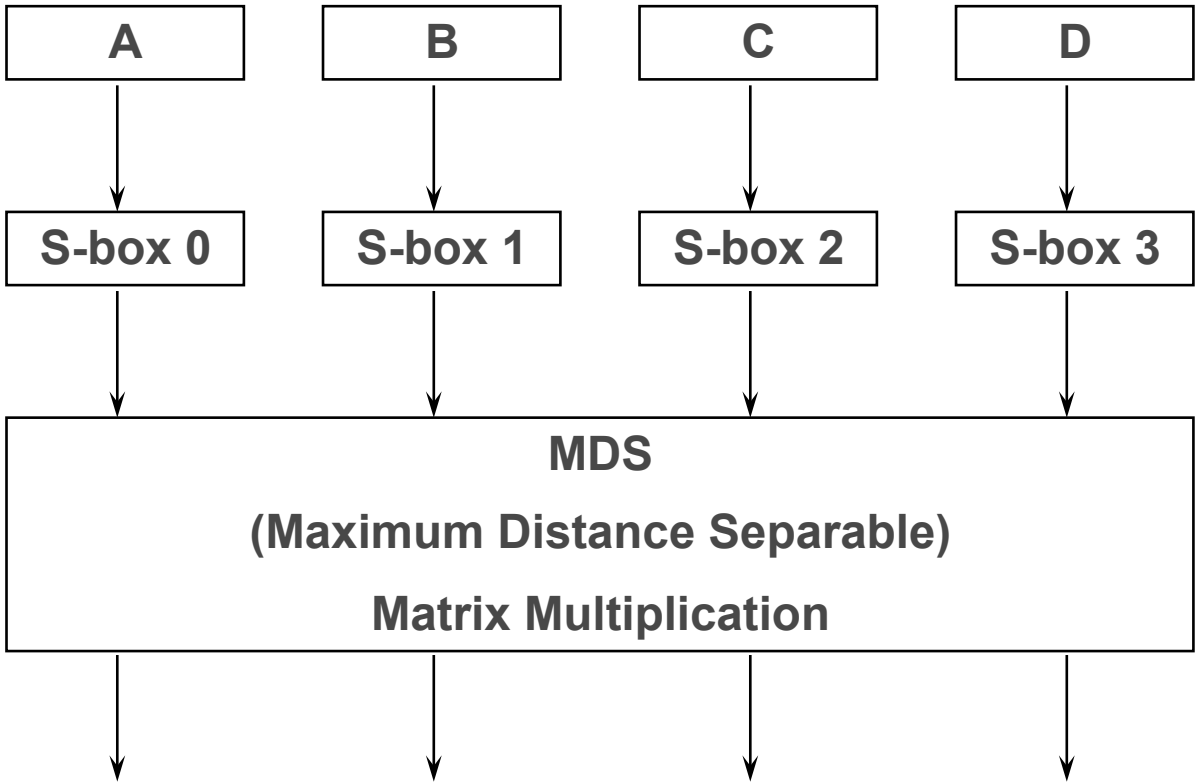
# Twofish structure

16 rounds



QUALCOMM

# Twofish *g* function



QUALCOMM

# 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**

QUALCOMM

# MARS

- Lots of authors from IBM
- Type 3 Feistel Network
  - one word modifies other three in each round
- Cryptographic core protected by outer, unkeyed, “mixing rounds”
- Expanding S-box and data dependent rotation
- Uses multiplication, so slow on smart cards
- Keys can’t be computed “on the fly”
  - Key schedule was “tweaked” to allow calculation in four lumps

QUALCOMM

# MARS Keying

- Accepts 4..14 words (128-448 bits)
  - pads, then adds a 15th word which is the key length
- Three stages:
  - LFSR-like filling of key array
  - repeatedly stir key words through the S-Box
  - “fix” the ones used for multiplication
    - Set least significant two bits
    - correct long runs of 0’s or 1’s.
- This is very complicated and slow

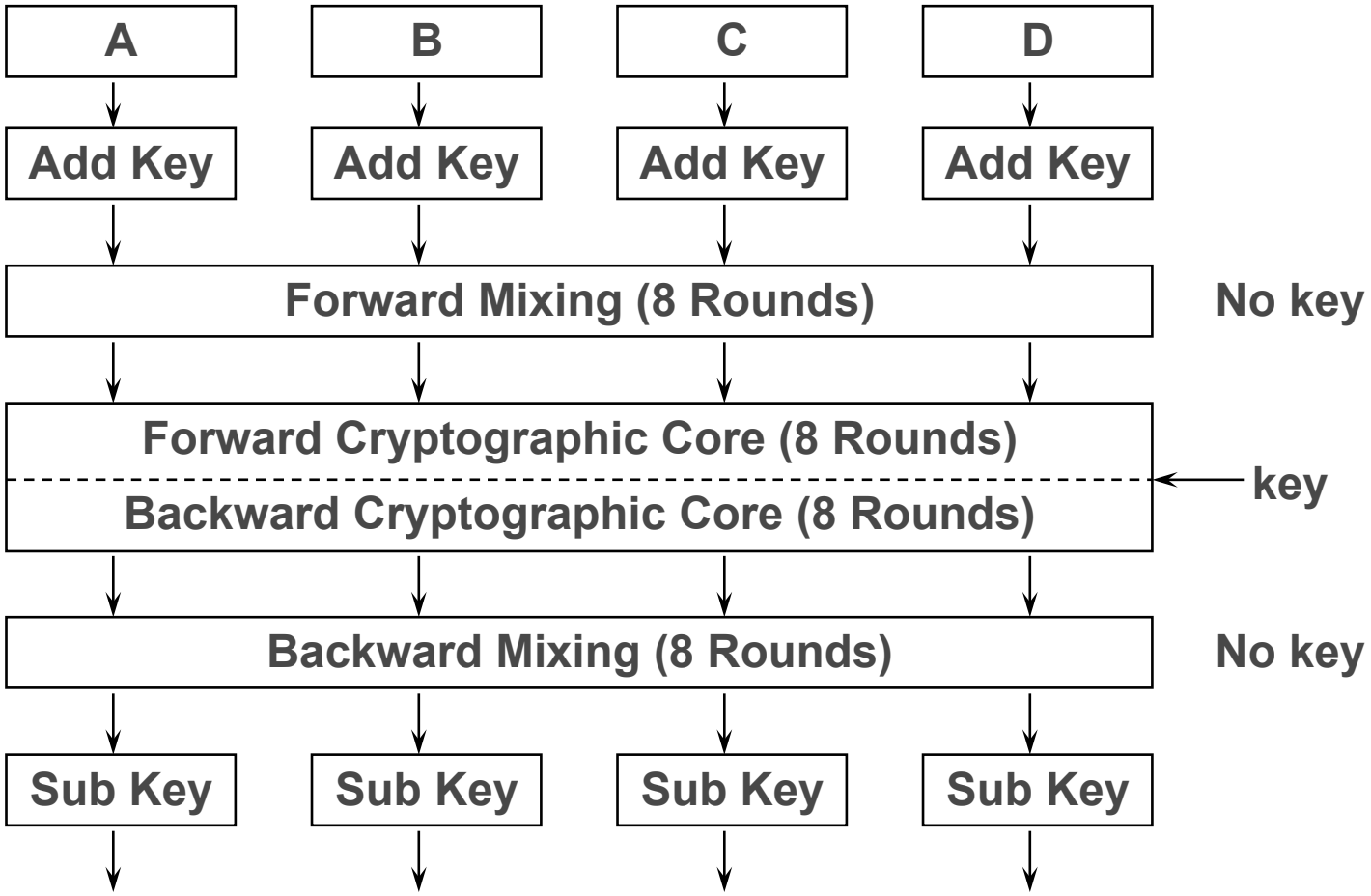
QUALCOMM

# 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.

QUALCOMM

# MARS Diagram



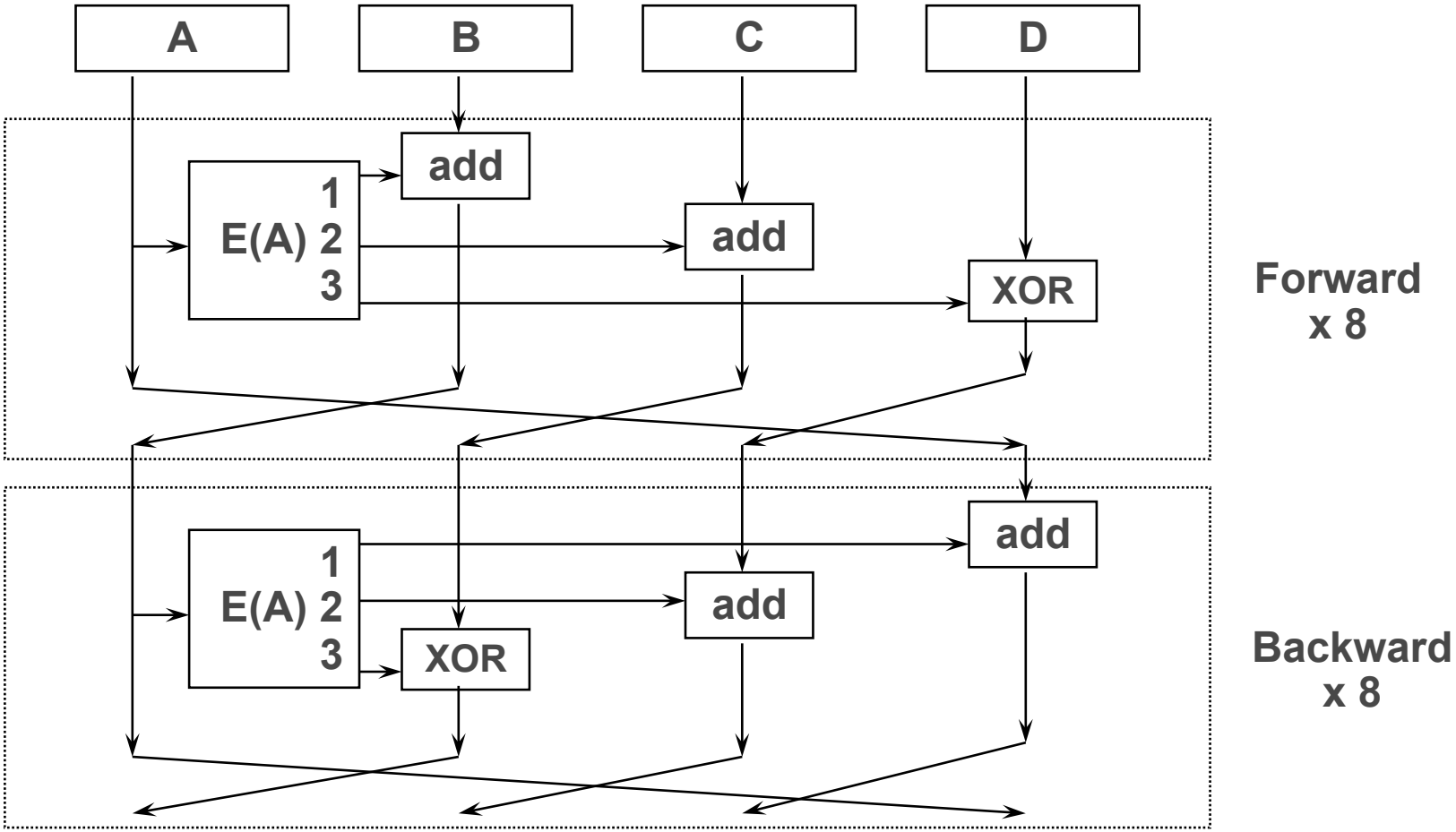
QUALCOMM

# MARS rounds

- Different functions of *A* used to modify *B, C, D*
  - in backward mode, used to modify *D, C, B*
- Words are rotated, *(A, B, C, D) = (B, C, D, A);*
  - same rotation whether forward or backward
- Backward mixing rounds are sort-of inverse operations of forward (see below)
- See paper for details of mixing rounds.

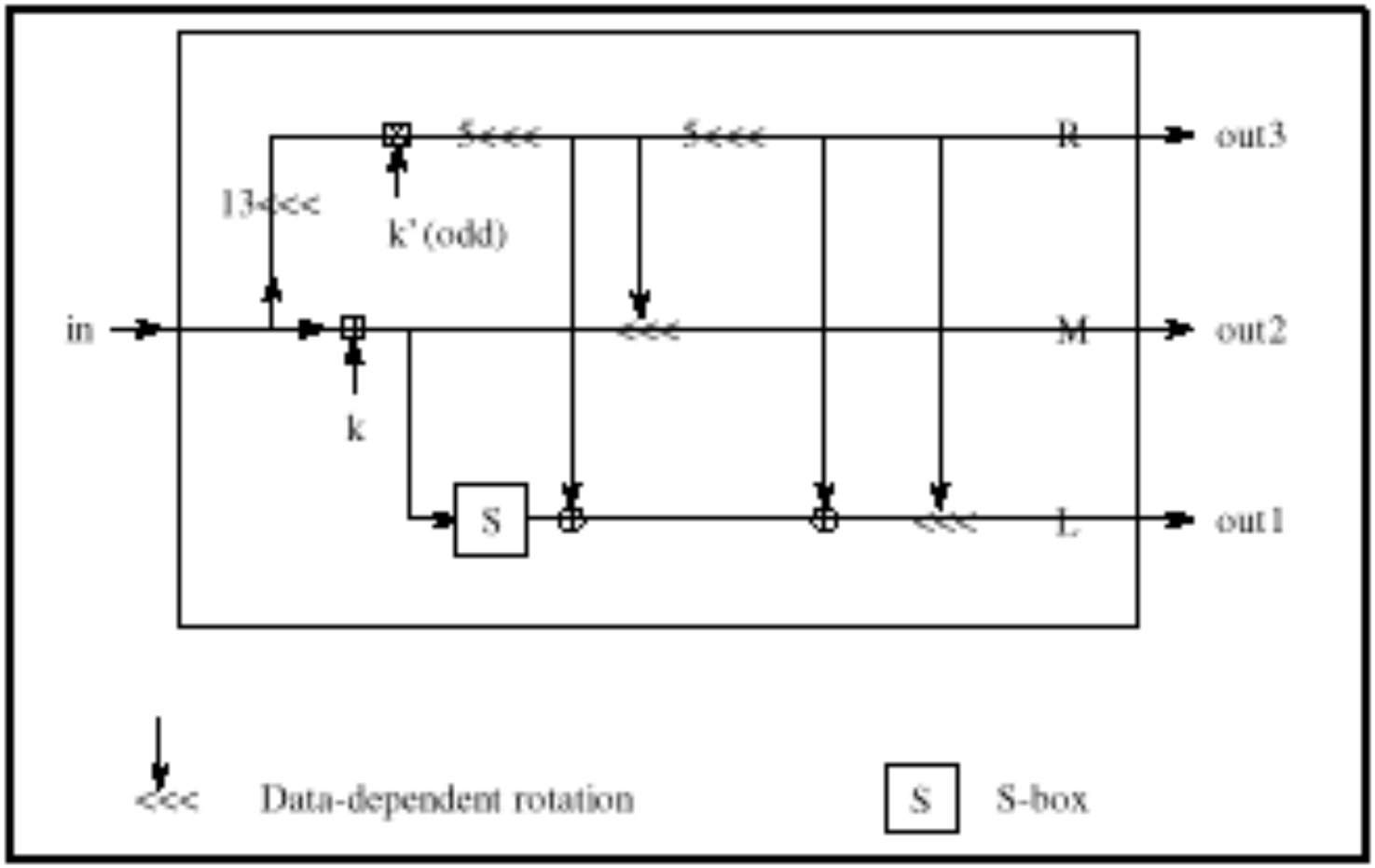
QUALCOMM

# MARS Core Rounds



QUALCOMM

# MARS E-function



QUALCOMM

# 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.

QUALCOMM

# My guess about the others

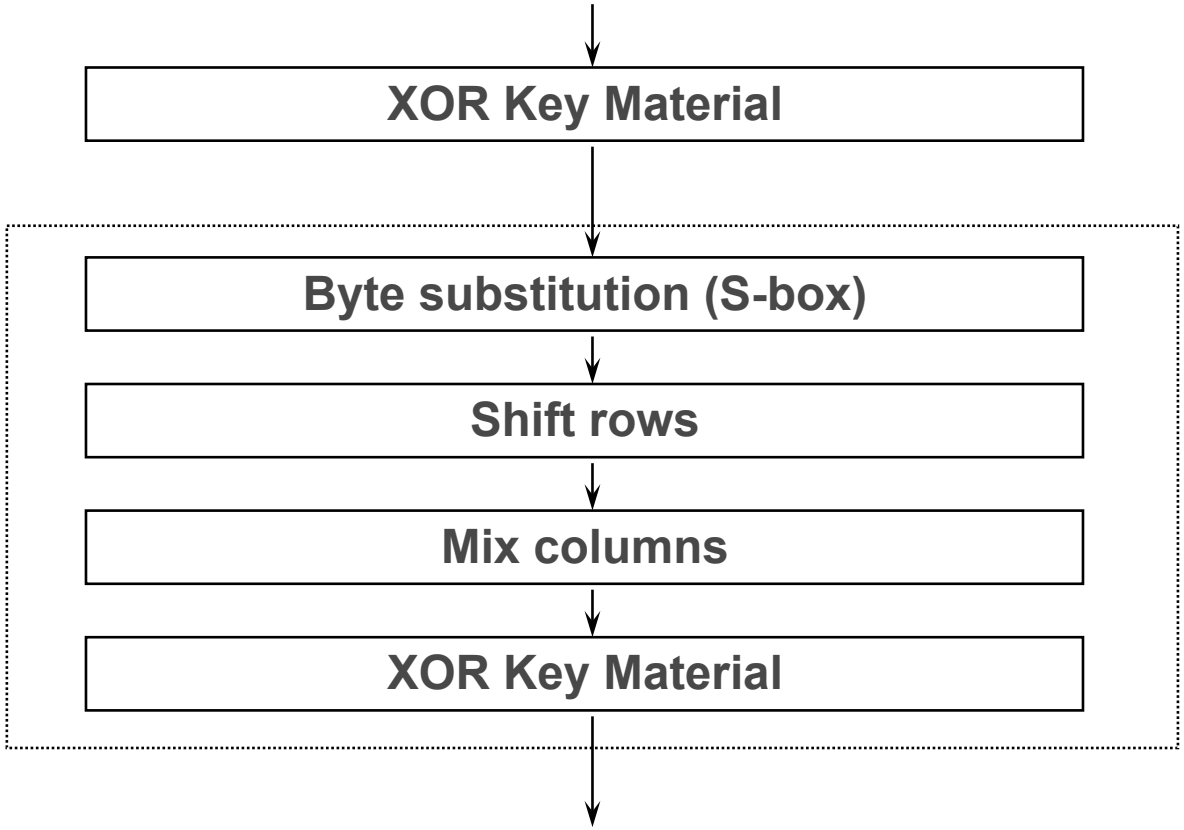
- RC6 requires fast multiply, expensive hardware
- MARS also needs multiply, and memory for key schedule
- Twofish had a “key separation” doubt
- Serpent too slow because of too many rounds (not their fault) – probably the second choice

QUALCOMM

# 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)

# Rijndael structure



QUALCOMM

# Rijndael keyschedule

- Similar to Serpent's
- Use key 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)

QUALCOMM

# 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

# Rijndael ShiftRow

QUALCOMM

# 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
- This is an MDS transform

QUALCOMM

# 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

QUALCOMM

# Skipjack

- Skipjack, when declassified in 1998, was a revelation to the crypto community
- 64-bit blocksize, split into 16 bit words
- LFSR like structure with “forward” and “backward” rounds
- Extremely simple key schedule
- Feistel-like structure with 8-bit S-box on 16-bit words
- 32 simple rounds (f, b, f, b)

QUALCOMM

# Algebraic attacks

- Sometimes called “linearization” attacks
- Any cipher can be described as a multivariate high-degree polynomial
- Usually these are intractable
- … but if the degree isn’t too high…
  - Treat each monomial as a new variable
  - Solve equations as a linear system
- Attacks on
  - Serpent (S-boxes too small)
  - Rijndael (debatable) (algebraic structure in S-box)
  - Many, many stream ciphers.

QUALCOMM

# 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
- Also CTR mode

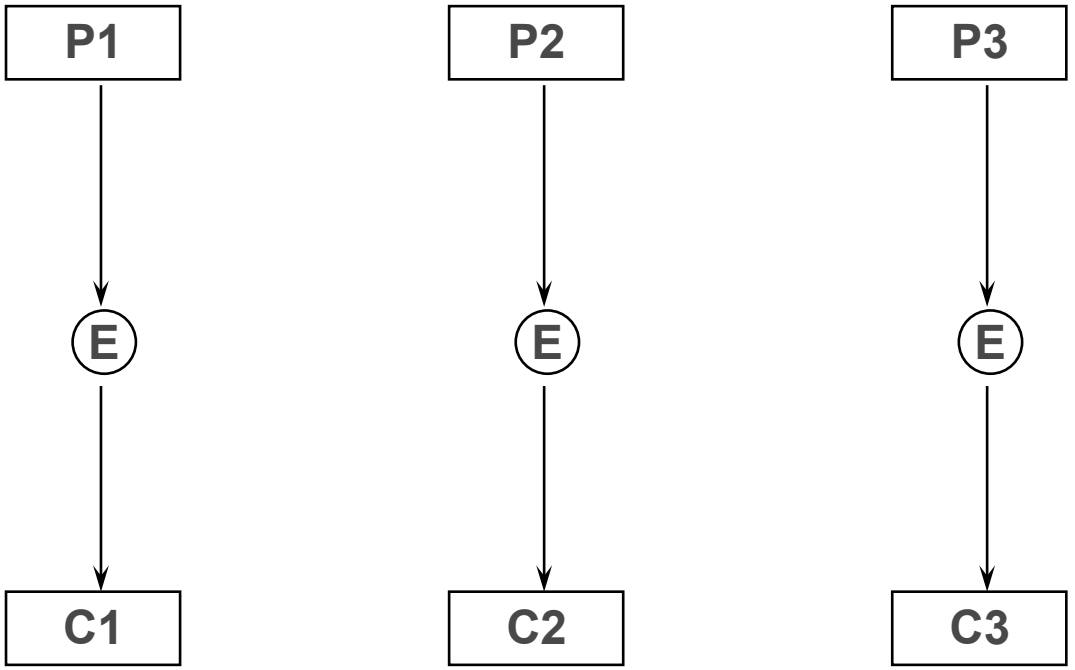
QUALCOMM

# 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!

QUALCOMM

# ECB Diagram

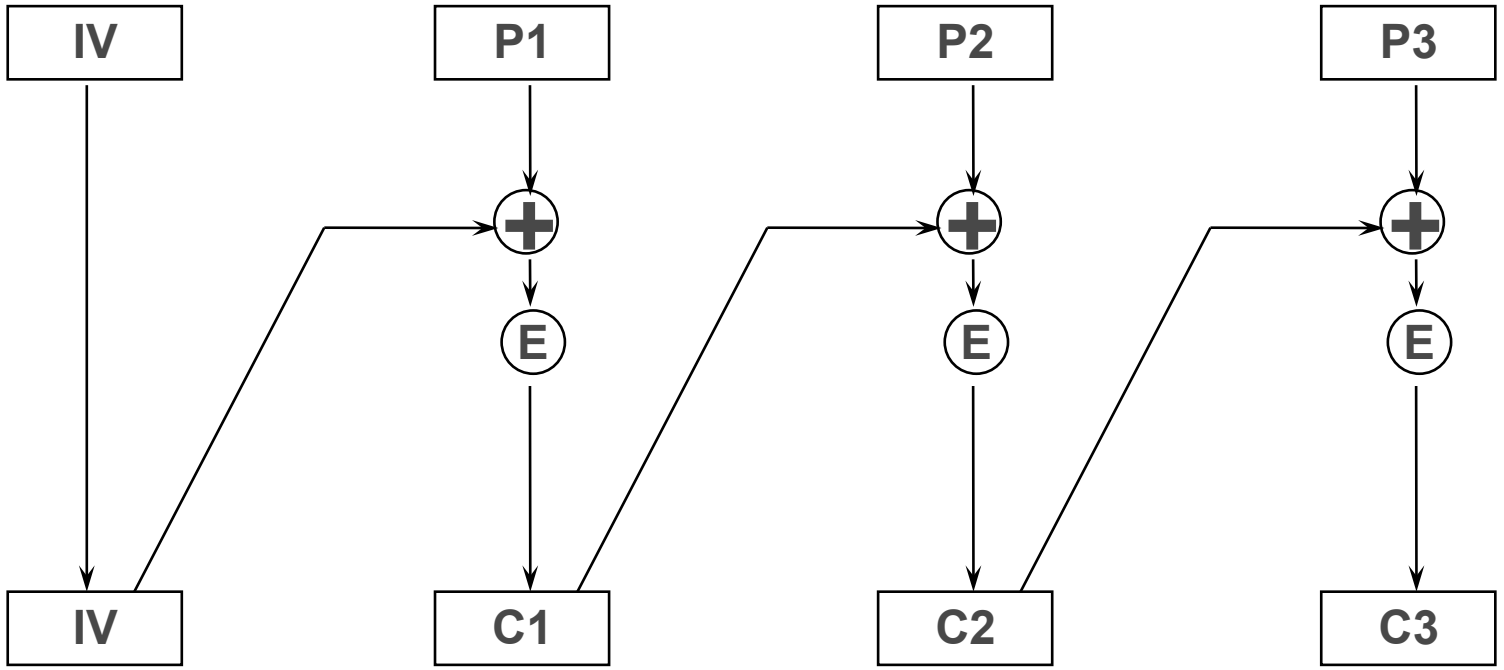


QUALCOMM

# 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

# CBC Diagram



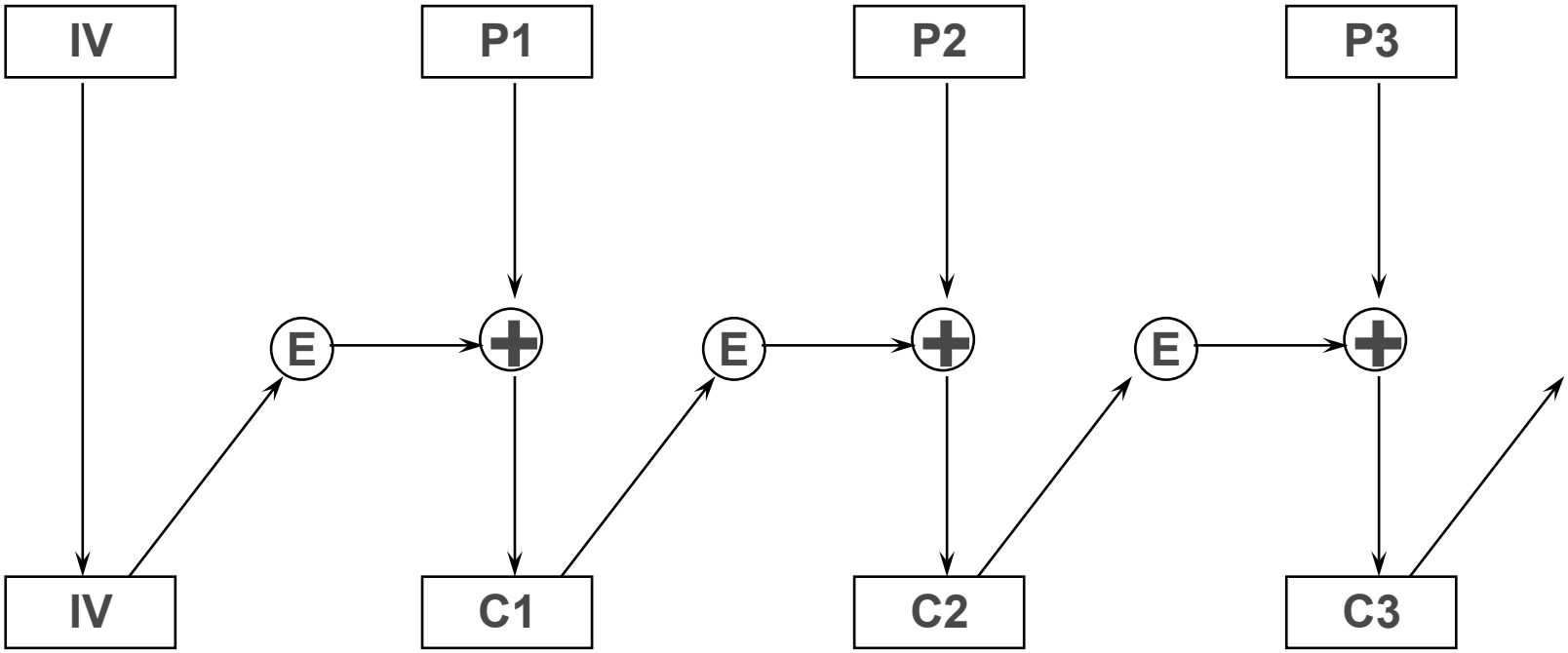
QUALCOMM

# 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.

QUALCOMM

# CFB Diagram



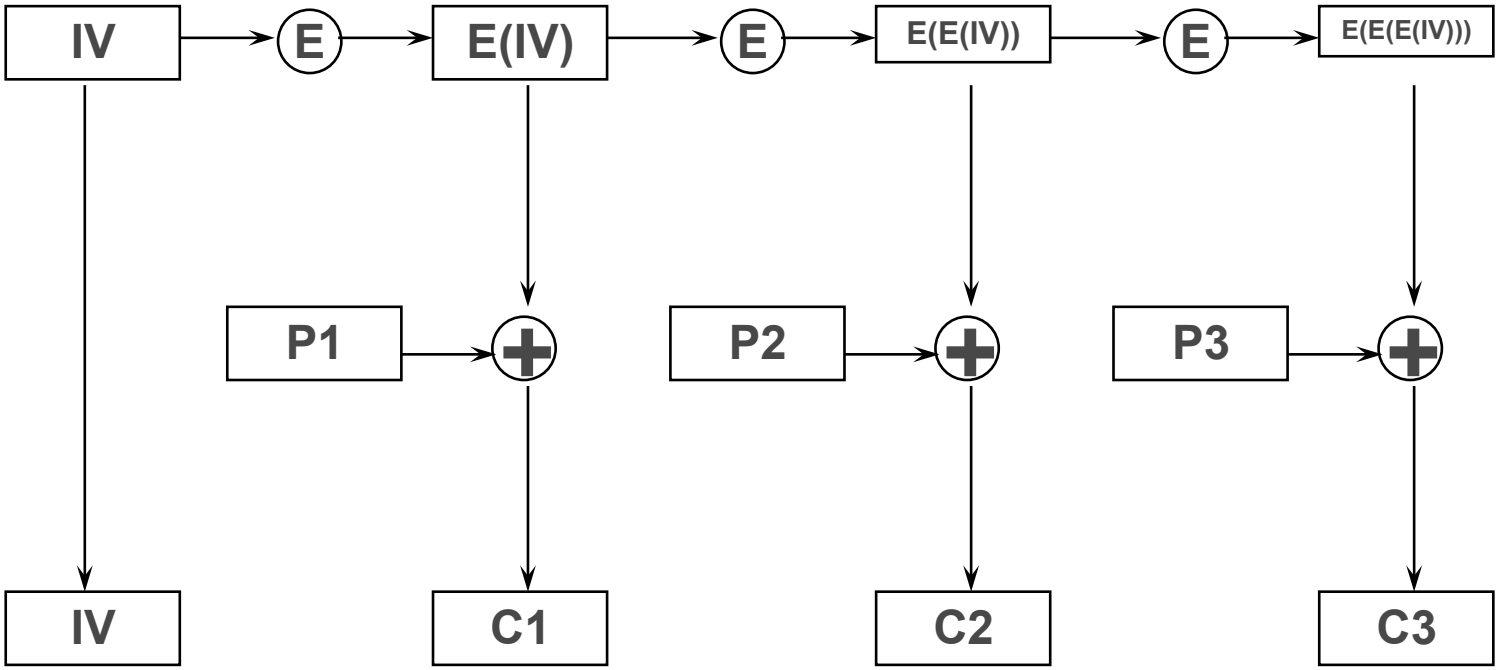
QUALCOMM

# Modes(4) - OFB and counter

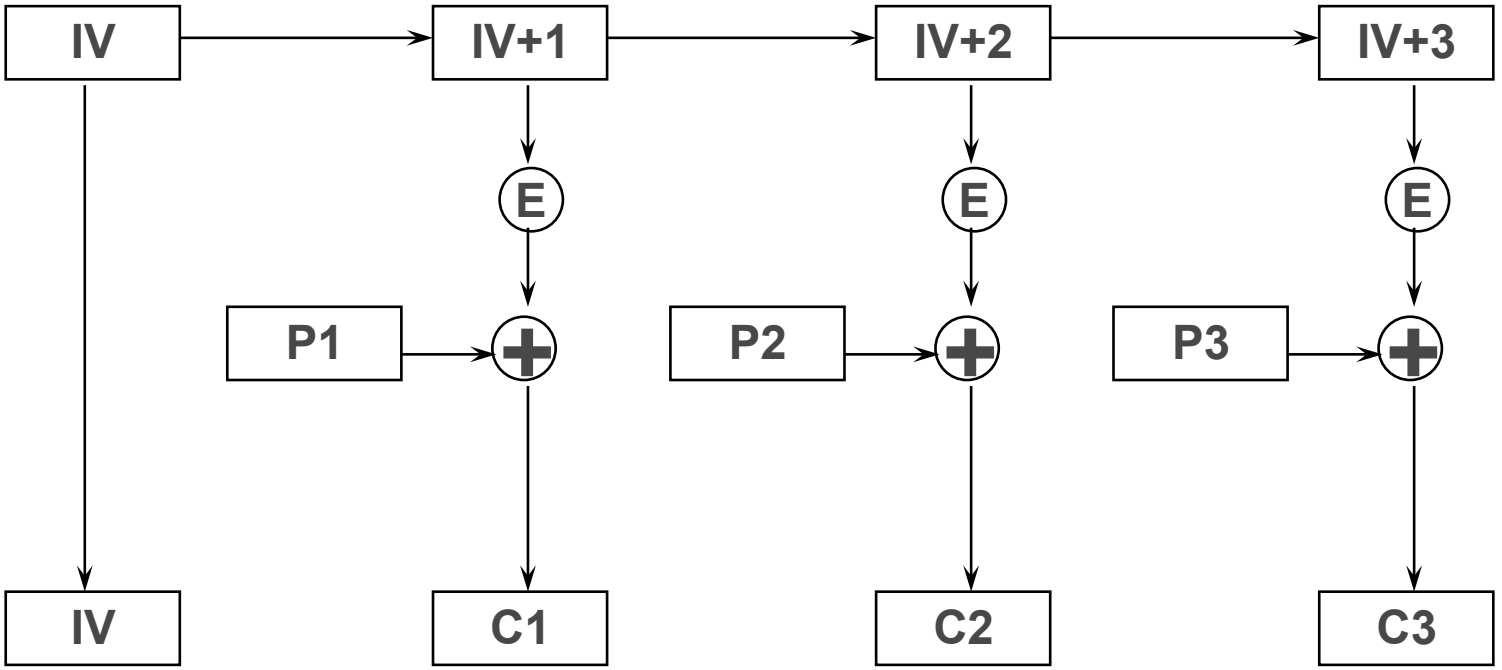
- Output FeedBack - Encrypt the previous output, then XOR with the plaintext to get ciphertext.
- Counter mode - start somewhere, and increment, encrypting at each stage, then XOR with the plaintext.
- These are effectively ways to turn a block cipher into a stream cipher
  - and have all the same problems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

QUALCOMM

# OFB Diagram



# Counter Mode Diagram



QUALCOMM

# New Modes of Operation

- NIST held a workshop in August 2001
- 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.
- Old slide – see FIPS special publication 800.

QUALCOMM

# 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

QUALCOMM

# 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.

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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.

QUALCOMM

# Fluhrer, Mantin, Shamir

- Some keys leave the state array “partially sorted”
- Have an identifiable bias in first byte of output
- Attack on related keys (CipherSabre, 802.11)
  - Gather lots of first bytes (~1 000 000)
  - Try all values for last byte of key
  - Correct guess can be checked statistically
  - Iterate – linear in key length.
- Note that discarding 256 bytes avoids attack

QUALCOMM

# 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*

QUALCOMM

# 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

QUALCOMM

# But now...

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

  **(gulp)**

  **… mathematics**

QUALCOMM

# 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:
  - $x^2 - x - 1 = 0$

QUALCOMM

# 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

QUALCOMM

# 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.*

QUALCOMM

# 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*.

QUALCOMM

# 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

QUALCOMM

# 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*.

QUALCOMM

# 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*

QUALCOMM

# 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-1)* over $F_2$ *do* form a field, $GF(2^n)$.

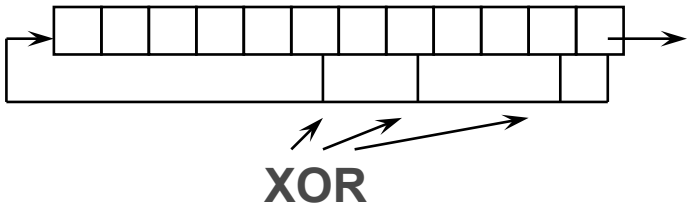
QUALCOMM

# 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

# 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
  - $C(x) = x^{12} + x^6 + x^4 + x + 1$

QUALCOMM

# 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

QUALCOMM

# 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**

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

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

QUALCOMM

# 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

QUALCOMM

# 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.**

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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.

QUALCOMM

# SOBER

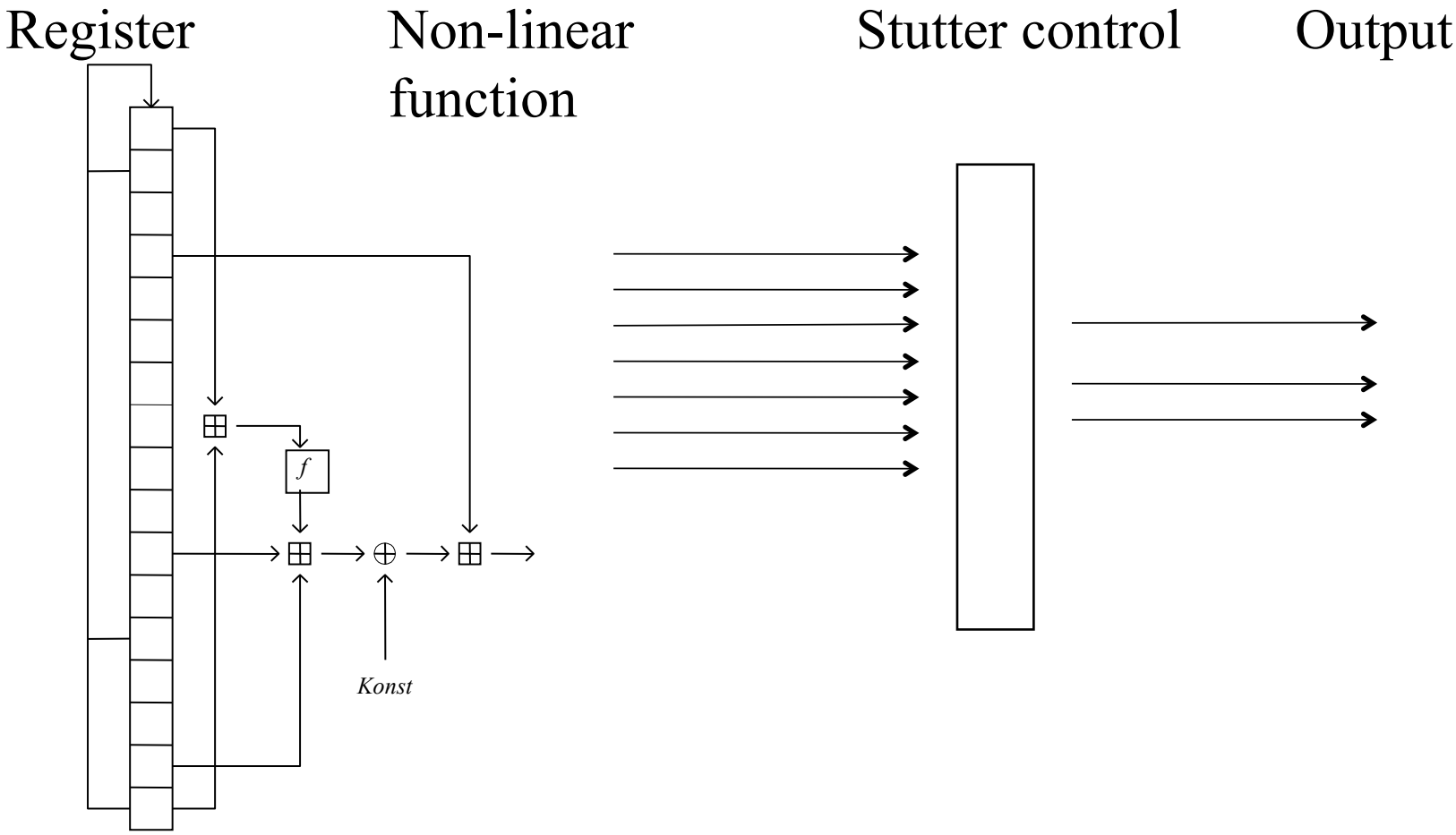
- Greg Rose (yes, me)
- uses LFSRs but with nice size chunks for software
- three versions, for different word lengths
  - t8, t16, t32
- fast keying, small memory, fast generation
- free for non-embedded use

QUALCOMM

# LFSRs over GF($2^8$)

- Elements of field are byte-sized binary polynomials
- Addition operation is XOR, $\oplus$
- Multiplication is poly-mod multiplication, $\otimes$
  - 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

# SOBER block diagram



QUALCOMM

# The Shift Register

- Generates maximal length sequence
  - period $2^{136} - 1$
- Recurrence:
  - $s_{n+17} = 99 \otimes s_{n+15} \oplus s_{n+4} \oplus 206 \otimes s_n$
- 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”

QUALCOMM

# 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

QUALCOMM

# The Stuttering

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

| | |
|---|---|
| **00** | **No output** |
| **01** | **Output *v* ⊕ *0x69***<br>**Cycle register again** |
| **10** | **Cycle register again**<br>**Output *v*** |
| **11** | **Output *v* ⊕ *0x96*** |

QUALCOMM

# 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.

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# Panama

- Joan Daemen and Craig Clapp
- billed as “cryptographic primitive”
  - a stream cipher
  - hash function
- *push* and *pull* functions
  - push stuff in (key or data to be hashed)
  - cycle a few times
  - pull stuff out (stream or hash value)
- Very fast generation, very slow initialization
- Broken.

QUALCOMM

# More Panama

- Large state
  - 32 bit wide parallel 17-bit LFSR
  - 16 x 8 word x 32 bit accumulator, sortof like another LFSR
- nonlinear transfers between LFSR and Accumulator
- Too complicated to describe here
- Attacks have been identified

QUALCOMM

# 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

QUALCOMM

# 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

QUALCOMM

# 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*

QUALCOMM

# 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

QUALCOMM

# 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*

QUALCOMM

# 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*

QUALCOMM

# 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*...

QUALCOMM

# 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.

QUALCOMM

# 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

QUALCOMM

# 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*

# Discrete Log encryption and digital signatures

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

QUALCOMM

# 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

QUALCOMM

# 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 SHA-1 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* | *(P-1)*
- Choose *g* to be a generator of the order-*q* subgroup
- Choose *x* < q; Public key is $(P, q, g, y = g^x)$

QUALCOMM

# 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.

QUALCOMM

# 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**

QUALCOMM

# 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.

QUALCOMM

# 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

QUALCOMM

# 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 mod p, b mod q*) to (*c mod N*)**
- **Note that (*1 mod p, 1 mod q*) is (*1 mod 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**

QUALCOMM

# Fermat applied to composites

- Given *M*, decompose to (*a mod p, b mod 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 $M^{(p-1)(q-1)} == 1 \bmod N$
- $M^{(p-1)(q-1)+1} == M \bmod N$

QUALCOMM

# 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, $C^d == M^{ed} == M^{(p-1)(q-1)+1} == M^1 == M$

QUALCOMM

# 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)

# 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".

QUALCOMM

# 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.

QUALCOMM

# 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”

QUALCOMM

# 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

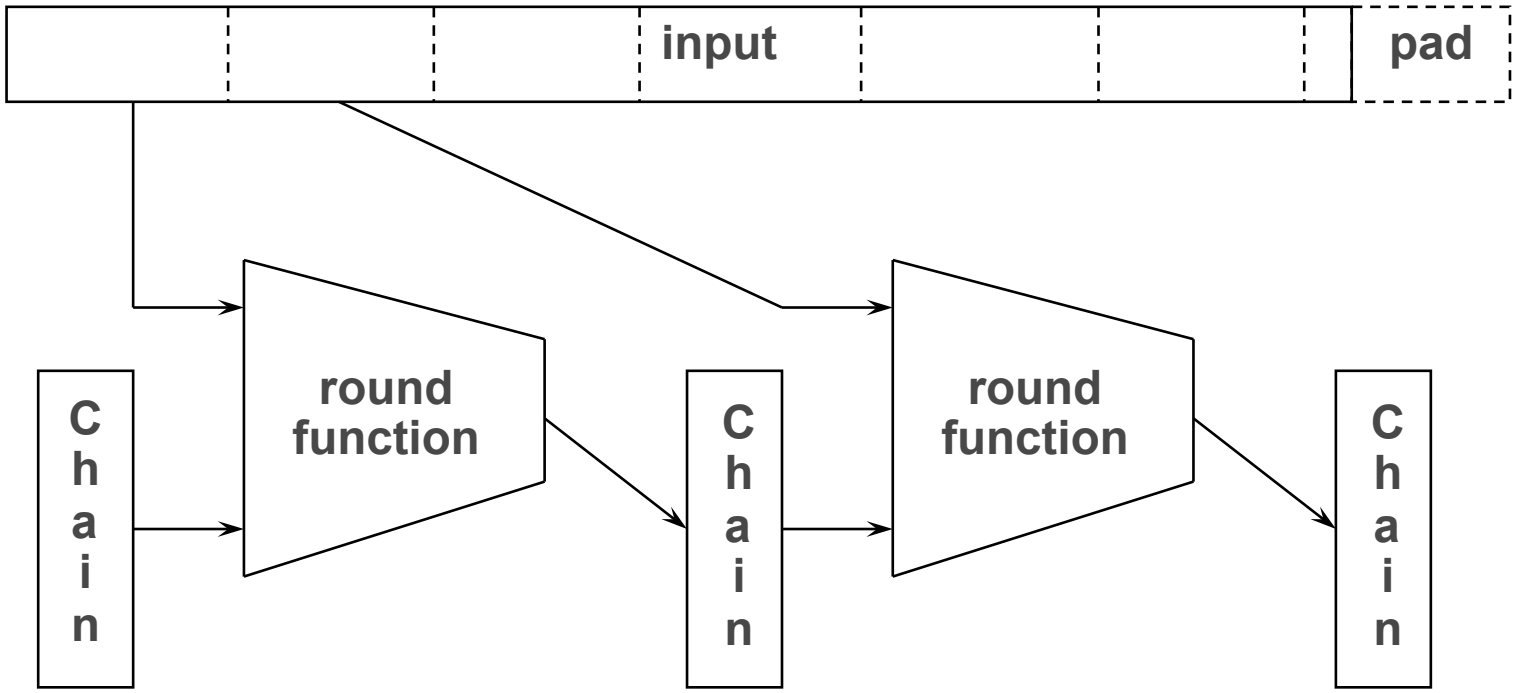
QUALCOMM

# 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

QUALCOMM

# Hash structure diagram



QUALCOMM

# 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)

QUALCOMM

# 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***

QUALCOMM

# 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).

QUALCOMM

# 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

QUALCOMM

# Keyed MAC -- HMAC

- Turning hash into MAC sounds easy, but isn't
  - Extension attacks
- HMAC based on NMAC
  - NMAC "provably secure in random oracle"
- MAC=H(k ⊕ opad, H(k ⊕ ipad, data))
- opad and ipad just simulate different hash functions
- MD5 has problem but HMAC-md5 avoids it

QUALCOMM

# What is Message Authentication?

- Also called “message integrity”
- Provides assurance that the message:
  - Came from who you thought it came from
  - Wasn’t changed along the way.

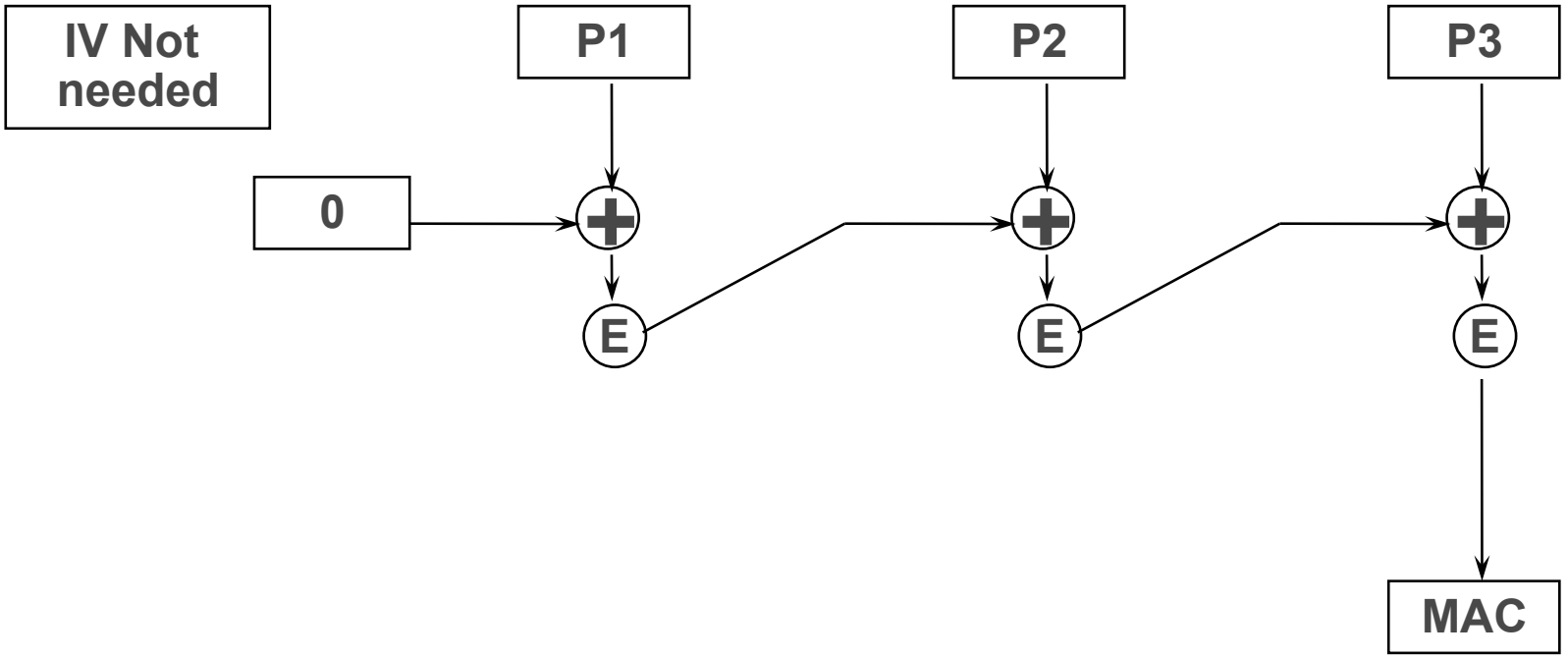
- Usually done by calculating a “keyed hash”, called a MAC (Message Authentication Code)
  - Sometimes called “tag”

QUALCOMM

# Nonces versus IVs

- Nonces are nice:
  - Can be smaller than a whole block
  - Only requirement is uniqueness, that is, never use the same one twice
  - Can be a timestamp
  - Can be a counter
  - Can be derived from the LSBs of a small counter
  - Can be derived from the environment somehow
- IVs are not so easy:
  - Must be pseudorandom and unpredictable

# CBC-MAC Diagram



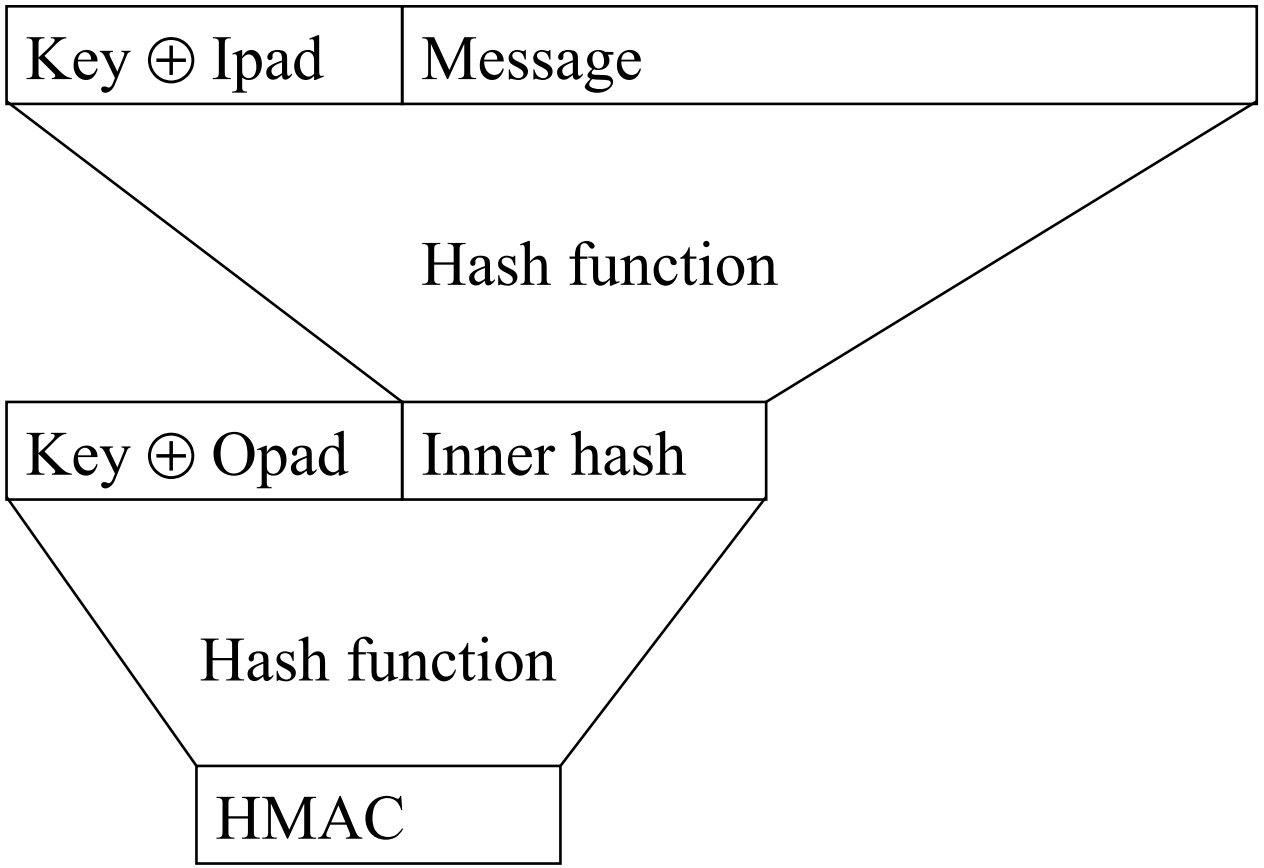
QUALCOMM

# Keyed MAC -- HMAC

- Turning hash into MAC sounds easy, but isn't
  - Extension attacks
- HMAC based on NMAC
  - NMAC "provably secure"
- MAC=H(k ⊕ opad, H(k ⊕ ipad, data))
- opad and ipad just simulate different hash functions
- MD5 has problem but HMAC-md5 avoids it

QUALCOMM

# HMAC diagram



# Why combine Encryption and Message Authentication?

- Some people think that
  - because you can't understand the message…
  - and if you modify it you don't know what you'll get
  - … encryption prevents mucking with messages
- *Wrong!*
  - Bellovin's cut-and-paste attacks on IPsec
  - early ATM attack in England
- More recently…

QUALCOMM

# Why? (cont)

- Some people think that:
  - because you can't decrypt the data …
  - … encryption alone provides data privacy.
- *WRONG!*
  - Bellare and Namprempre's "Encrypt then Authenticate" paper, and Krawczyk
  - Some of Bellovin's examples
  - 802.11 WEP (see Borisov, Goldberg & Wagner)
  - all are examples where modifying the message allows you to con someone else into decrypting it for you

QUALCOMM

# Still Why?

- **To get data confidentiality in the face of an active attack:**

  ***You must also have message integrity!***

- **So, you have to have it. Why a combined mechanism?**
  - **Too easy to make mistakes with cobbled-together solutions**
  - **… again, *Encrypt then Authenticate***
  - **possible efficiency gains**

QUALCOMM

# Is Message Integrity enough?

- … of course not, or this slide wouldn't be here
- Something must prevent replay of valid messages!
  - Alice tells Bank to transfer $10 to Mal
  - Mal taps wire, then replays the message a million times
- "*Freshness*" is required
  - Something that varies but is checkable by recipient
  - Timestamp, sequence number, response to challenge, …
  - Not a cryptographic algorithm issue

QUALCOMM

# Enough yet?

- No, attacker can always prevent delivery of the message
- Don't design protocol with "Are you sure?" messages with back consequences
- "Abort countdown?" even worse

QUALCOMM

# Enough why, now *what*?

- Think of IPsec
- want to have message integrity for whole packet
  - including source, destination, etc.
- but only want to encrypt the payload
  - otherwise delivering it gets tricky
- Want a flexible mechanism that allows message integrity over both plaintext headers and ciphertext
- New name: "AEAD", Authenticated Encryption with Associated Data (Rogaway and Wagner)

QUALCOMM

# Early attempts

- Block ciphers give "a bit of message integrity"
  - if two fields fall in one block, hard to modify one without modifying the other
- Some chaining modes propagate errors
  - same as above, just a bit bigger
- Encrypt with CBC, MAC with CBC-MAC
  - each part is secure
  - put them together, becomes insecure
- "The Holy Grail" for a while
  - Many thought it was impossible to do "one pass"

QUALCOMM

# Block Cipher based (1)

- Suddenly, in 2001:
  - IAPM (Jutla, IBM)
  - XCBX (Gligor & Donescu)
  - OCB (Rogaway et. al.)
- The above encrypt/authenticate whole messages
  - HR mode (Hawkes & Rose) extends to Partial Encryption with Message Integrity
- High throughput parallelizable modes
- All patented

QUALCOMM

# How do they work?

- All similar in concept
- All are proven correct based on reasonable assumptions about the security of the underlying block cipher
- combines a secret counter with plaintext, then encrypts
- MAC is a checksum of intermediate results, then encrypted itself
- (This is a gross oversimplification!)

QUALCOMM

# Block Cipher based (2)

- Because of patents, much of the community has backed off from use of those modes.
- Instead, go back to using block cipher in counter mode for encryption
- For message integrity, use something either:
  - Based on the block cipher, for speed and commonality with the encryption, or…
  - Use a very fast universal hash function then encrypt it.
- AES-CBC + HMAC-SHA-1 (ie. IPsec, TLS) is secure, just too slow.
  - also can't be parallelized in any way

QUALCOMM

# New "combined" modes

- CCM
- EAX
- CWC

- Of course there's the old, trusty, combination of CBC mode encryption and HMAC

QUALCOMM

# CCM

- Whiting, Ferguson and Housley
- "Counter with CBC-MAC"
- Encrypts with counter mode
- Uses CBC-MAC for Message Authentication Code
- Used in 802.11 next generation security
- Prepends header with length of message
  - makes it hard to process a stream
- Very "bit fiddly"

QUALCOMM

# EAX

- Bellare, Rogaway and Wagner
- Encrypt then Authenticate then X(trans)late
- Encryption is Counter Mode
- MAC is just a little more complicated
  - based on CBC-MAC variant called OMAC
  - length field gets mixed in at the end
  - complicated padding to avoid extending data

- (Coined the term AEAD)

QUALCOMM

# CWC

- Kohno, Viega, Whiting
- "Carter-Wegman and Counter"
- Encryption is same old Counter Mode
- MAC is Carter-Wegman construct:
  - treat the input data as 96-bit coefficients of a polynomial
  - evaluate polynomial with x = key
  - all modulo 2^127-1 (prime, easy arithmetic)
  - encrypt the result
- Potentially faster than the others
  - big multiplications a bit faster than encryption

QUALCOMM

# Stream Ciphers

- Note that all of the above use Counter Mode for the bulk encryption. This just makes the block cipher into a stream cipher
- But stream ciphers can be significantly more efficient!
- New rules required:
  - Never allowed to re-use nonces
  - MUST discard messages failing authentication, which sounds obvious, but some people try to use them.
  - "A bit in the message says whether MAC is required or optional…"

QUALCOMM

# Helix

- Ferguson, Whiting, Schneier, Kelsey, Lucks, Kohno
- 5 state words, combined using shifts, addition and XOR in a helical pattern
- Each block produces one word of keystream, and accepts one word of plaintext for MAC
- At end of block, run it for a little while longer, then generate keystream for MAC.

QUALCOMM

# SOBER-128

- Rose & Hawkes
- Based on earlier SOBER ciphers, particularly SOBER–t32.
- MAC aspect similar to Helix
- 17-word LFSR state
- 5 word nonlinear function, 1 word output
- plaintext word added into LFSR using nonlinear function
- Generate MAC by mixing some more then output keystream

QUALCOMM

# Performance

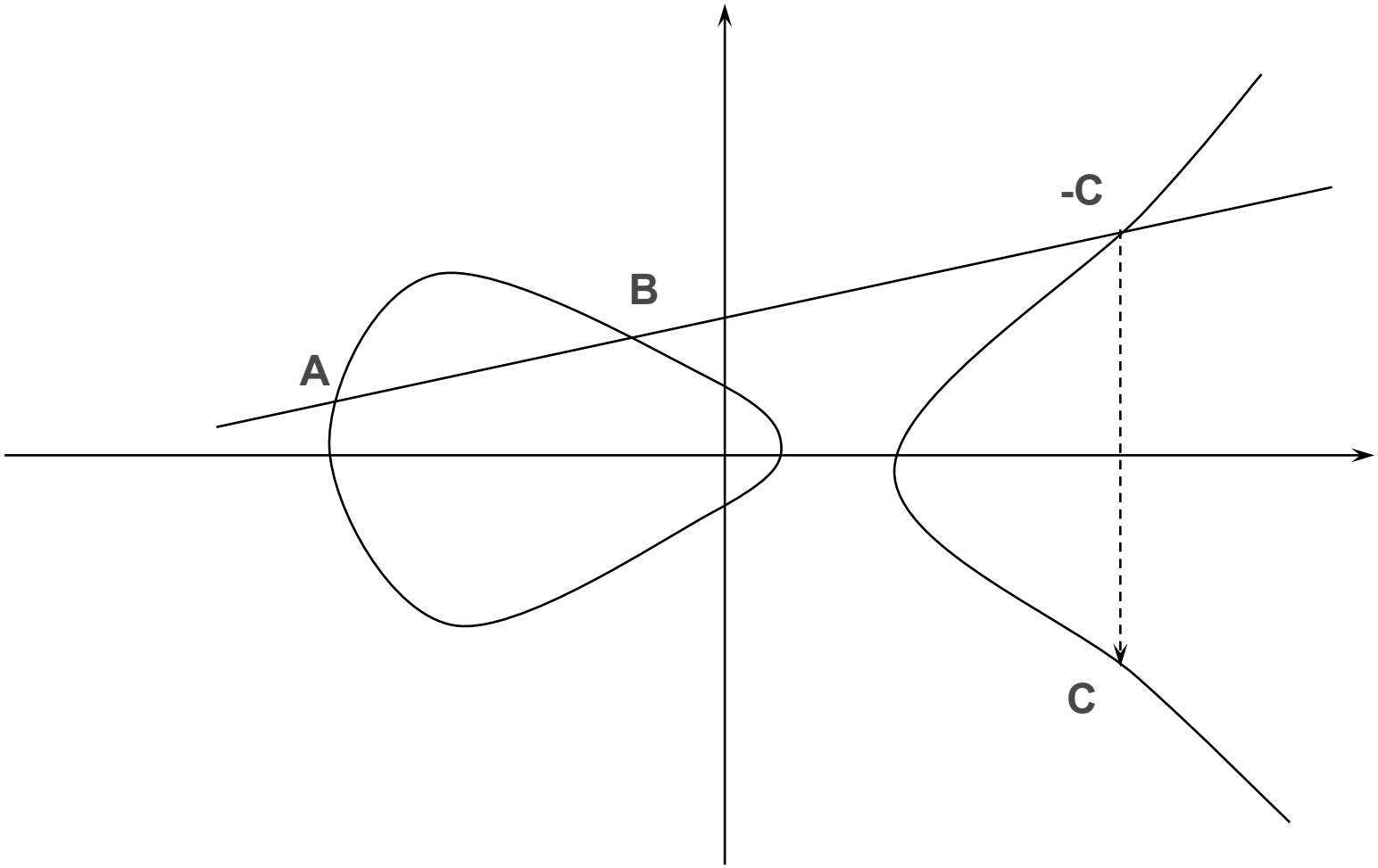
- Brian Gladman's figures, Pentium 4:
  - CCM: 28 * header_length + 48 * message_length
  - CWC: 40 * header_length + 66 * message_length
  - EAX: 23 * header_length + 48 * message_length
- My figures, Centrino @ 1.5Ghz:
  - Helix: 30 cycles per byte (49 MB/s)
  - SOBER-128: 8.6 cycles per byte (175 MB/s)
- Not strictly comparable!

QUALCOMM

# 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

# Curve over the real numbers



QUALCOMM

# 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)

QUALCOMM

# 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

QUALCOMM

# 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