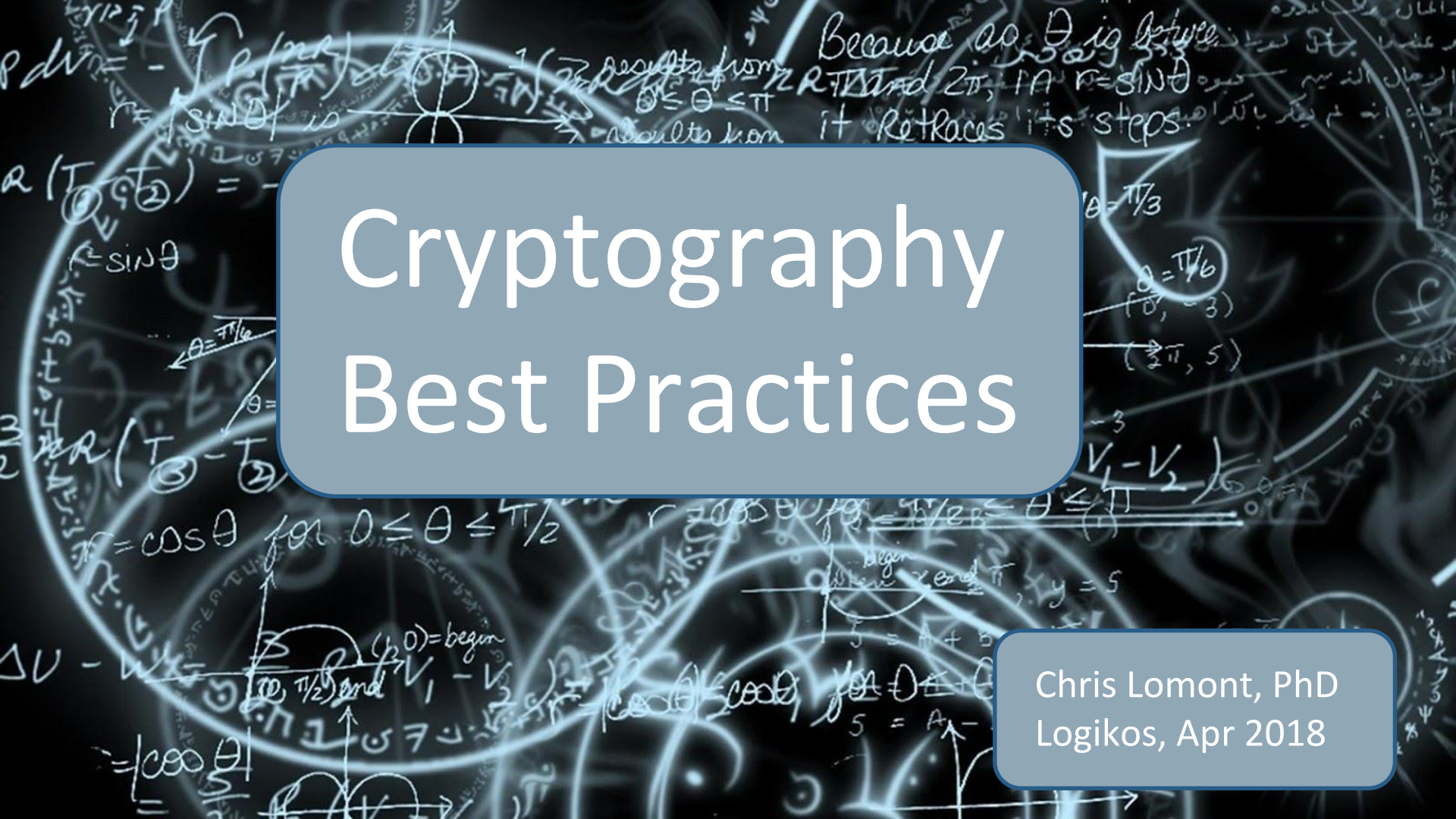


Cryptography Best Practices

Chris Lomont, PhD
Logikos, Apr 2018

HELLO!

I am Chris Lomont

You can find me at
clomont@logikos.com
Chris@Lomont.org

About Logikos

- Software Services
  - Fort Wayne, IN
  - Founded 1978
- Areas
  - Embedded
  - Desktop
  - Web Applications and Design
  - Mobile Applications
  - Software Testing
- Clients
  - Automotive
  - Commercial Services
  - Consumer Electronics
  - Govt and DoD
  - Medical
  - Retail
  - Others

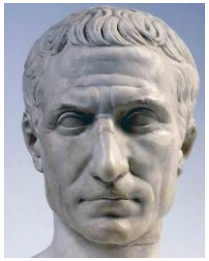
Encryption

Cryptography (*noun)*

- the practice and study of techniques for secure communication in the presence of adversaries

“Few false ideas have more firmly gripped the minds of so many intelligent men than the one that, if they just tried, they could invent a cipher that no one could break.”
-- David Kahn, in "The Codebreakers"

Substitution ciphers



- Caesar cipher (used by Julius Caesar with a shift of 3)

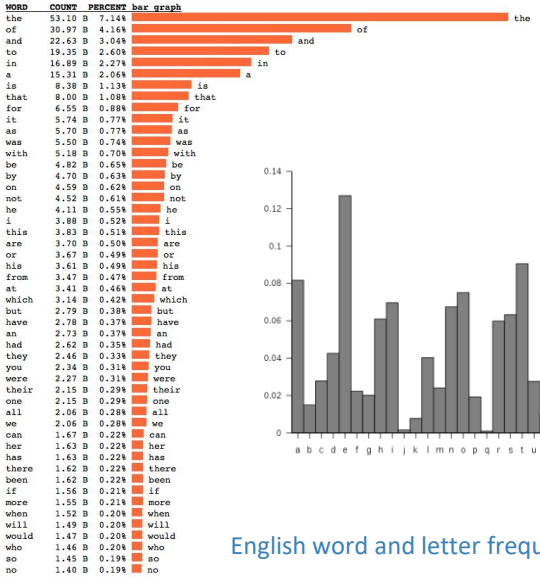
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- Encrypt: substitute letters in row 2 for letters in row 1
  - *Encrypt*(**COMPUTER**) gives **FRPSXWHU**

- Decrypt: substitute letters in row 2 for letters in row 1
  - *Decrypt*(*Encrypt*(**COMPUTER**))= *Decrypt*(**FRPSXWHU**) = **COMPUTER**

Breaking simple ciphers

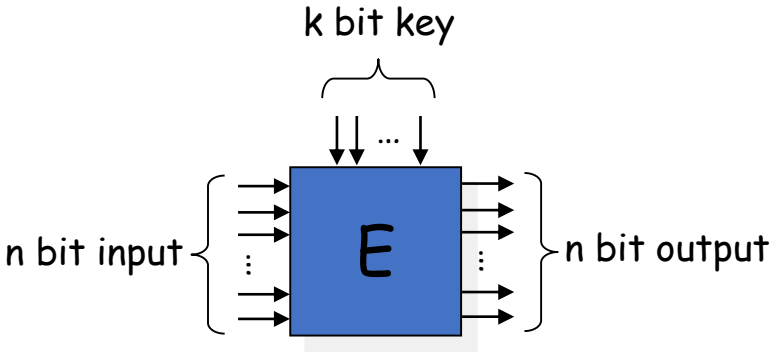
- Broken by frequency analysis
  - ‘e’ and ‘t’ most common letters
  - Common 2 and 3 letter words give more
- First known recorded explanation of frequency analysis
  - 9th century, “*A Manuscript on Deciphering Cryptographic* by Arab Al-Kindi
  - First writing of any kind of cryptanalysis



English word and letter frequencies

Block ciphers

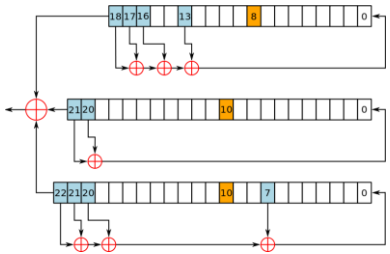
- an *n bit block cipher* is a function E: $\{0, 1\}^n$ x $\{0, 1\}^k \rightarrow \{0, 1\}^n$, such that for each key K $\in \{0, 1\}^k$, E(x, K) is an invertible mapping from $\{0, 1\}^n$ to $\{0, 1\}^n$



- DES, AES, hundreds more
- Message m broken into n bit chunks, each encrypted.
  - Last may need *padded*

Stream ciphers

- ***Similar to a one-time-pad***
- Alice creates a “stream” of “random” values,
  - Must be reproducible by listener Bob
  - XOR with message at both ends
- Examples
  - RC4 (Rivest, designed 1987, leaked 1994)
    - Bad usage led to break in WEP
    - 2015: speculation some state agencies can break when used with TLS
    - RFC 7465 prohibits use of RC4 in TLS
  - ISAAC (Jenkins 1993) – minor weaknesses, still ok to use
    - Used in UNIX **shred** to wipe data
  - Salsa20 (Bernstein, 2007)
    - Best known attacks break 8 of the 12 or 20 rounds, still usable
  - **ChaCha** (Bernstein 2008)
    - Selected by Google to replace RC4 in TLS
    - Used in OpenSSH, OpenBSD, NetBSD
    - Linux kernel uses for /dev/urandom



Keystream generator for A5/1, used to encrypt GSM mobile phones, weak/broken

Advanced Encryption Standard (AES)

- NIST sponsored competition, started 1997, needed to replace 3DES
- AES wins (Vincent Rijmen, Joan Daeman)
  - Federal Standard in 2002
  - Only publicly available cipher allowed for Top Secret data
- Modern, strong, fast
- Key is 128, 192, or 256 bits
  - 10, 12, or 14 cipher rounds
  - Lots of hardware support
- Attacks so far not very effective

Rounds	AES-128	AES-192	AES-256
8	125.4		
10	126.2		
12		189.7	
14			254.2

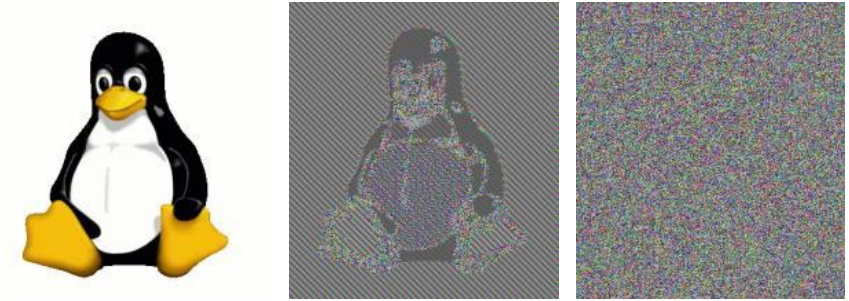
AES hardware and performance

- Hardware support common
  - Many servers spend significant time performing cryptography, speeds up greatly
  - Intel AES-NI on x86 chips most commonly used
    - 7 instructions, **AESENC** does one round
  - VIA PadLock
  - SPARC T3,4,5 and later
  - IBM Power7+
  - IBM z9 and later mainframes

- Example: Crypto++ (security library) AES-CGM
  - throughout from 28.0 -> 3.5 cycles per byte

Importance of chaining

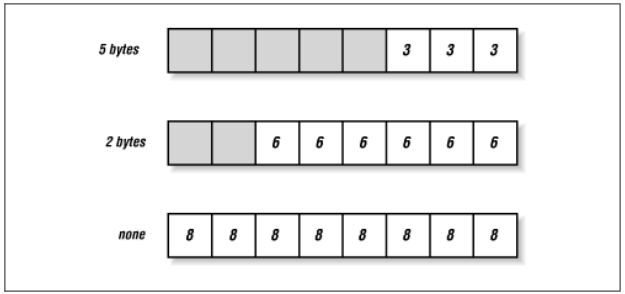
- Doing each block of message independently leads to this:



Example of DES on picture with no chaining (source Wikipedia)

- Leads to *chaining* one block to next during encryption/decryption
- Simple block by block called Electronic Codebook (ECB) chaining
- Common method is called Cipher Block Chaining (CBC, 1976)
  - AES-CBC is how you denote a block encryption algorithm and chaining mode

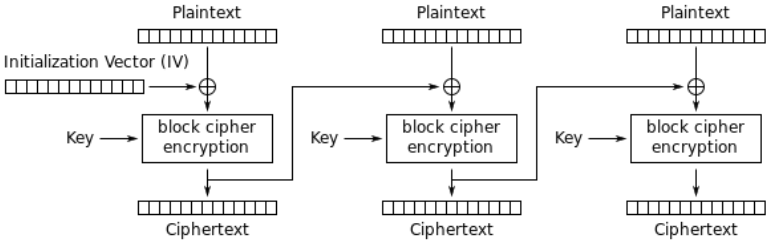
Padding modes



- Done incorrectly leaks information
  - OpenSSL prior to v0.9.6c had timing attacks in CBC-MAC
- Often done to make receiver able to remove padding easily
- Some methods
  - Single 1 bit, rest 0 bits (ISO/IEC 9797-1)
  - Last bit complemented and filled (Trailing Bit Complement)
  - PKCS#5 and PKCS#7: last byte is copy of number n of n bytes added
  - ISO 7816-4 Append 0x80, then all 0x00
  - ISO 10126-2: # of pad bytes, then random bytes
  - ANSI X9.23: 0x00 until last byte, then # of pad bytes
  - All 0's (fragile)

Cipher Block Chaining (CBC)

- XOR one block output to next block input
- First block requires *initialization vector* (IV)
  - Guards against repeated initial data attacks
  - *Must be generated securely, never reused*

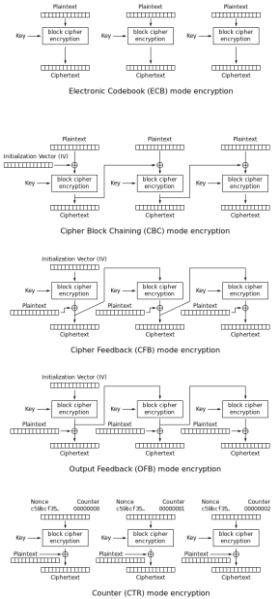


Cipher Block Chaining (CBC) mode encryption

- Most used chaining mode
- Drawbacks
  - Encryption must be sequential
  - Message needs padded

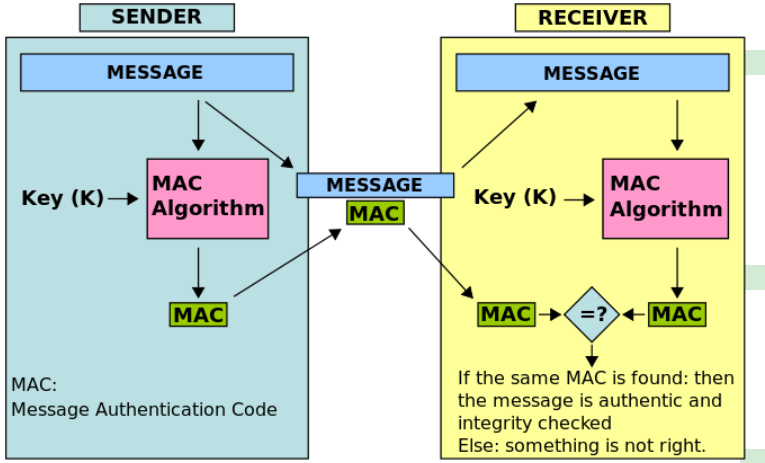
Common chaining modes

Electronic Codebook	ECB	Block by block (fast, parallel, weak)
Cipher Block Chaining	CBC	Cipher XOR with next plaintext (stronger, not parallel)
Cipher Feedback	CFB	Plaintext XOR into cipher blocks (strong, parallel, slower than OFB)
Output Feedback	OFB	Similar to CFB (encrypt/decrypt same, IV caution)
Counter	CTR	IV and counter, XOR plaintext (parallelizable, strong, IV caution)
Galois/Counter mode	***GCM***	***Encrypt counter, XOR plaintext (Fast, strong, gives integrity)***



Message Authentication Code (MAC)

- Also known as a “tag”
  A short piece of data used to ***authenticate*** a message
  - Authenticity: confirm the message came from the stated sender and has not been changed
  - MAC protects both a message’s ***data integrity*** as well as its ***authenticity***
  - Allowing verifiers to detect any changes to the message content using shared secret key

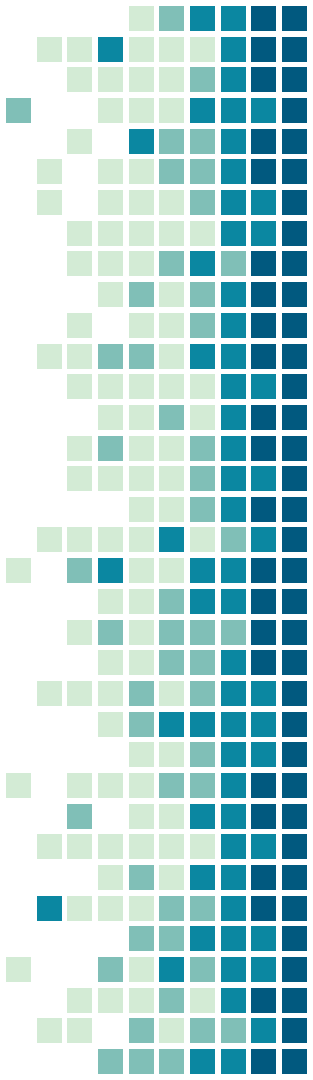


Poly1305

- Poly1305 (RFC 7539) : a modern message authentication code (MAC)

- Computes a 16-byte authenticator of a message of any length
  - 16-byte nonce (unique message number)
  - 32-byte secret key
  - breaking Poly1305 requires breaking AES

- Details:
  - message treated as 128 bit coefficients of a polynomial
    - over finite field field Z/(2^130−5)
  - excellent performance without precomputation
  - Important details in key format and message padding



NaCl / libsodium

- Networking and Cryptography Library: NaCl
  - pronounced “salt”
  - created by Daniel J. Bernstein
    - Creator of Chacha, Curve25519, Poly1305, much more
  - Main goal "avoid various types of cryptographic disasters suffered by previous cryptographic libraries".
  - C, bindings in many languages, very permissive license

Guidelines I: Symmetric Encryption

- Use authenticated Encryption (AEAD)
  - ChaCha20-Poly1305 is faster in software than AES-GCM.
  - AES-GCM will be faster than ChaCha20-Poly1305 with AES-NI.
  - Poly1305 is also easier than GCM for library designers to implement safely.
  - AES-GCM is the industry standard.
- Use: (ordered by preference):
  - The NaCl/libsodium default
  - Chacha20-Poly1305
  - AES-GCM
- Avoid:
  - AES-CBC, AES-CTR by itself
  - Block ciphers with 64-bit blocks such as Blowfish
  - OFB mode
  - RC4, which is comically broken

Symmetric/Asymmetric key cryptography

- Symmetric key (also called private key) methods – Alice and Bob have same, pre-shared key
  - Above methods like this
- Asymmetric key (also called public key) methods – Alice and Bob have different keys, usually not pre-shared
  - Diffie and Hellman's paper opened the door to a new kind of cryptography
  - Biggest advance in cryptography in 3000 year history

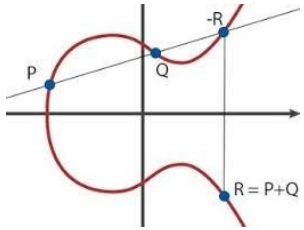
- Public key methods
  - Each user has a private key K as before, and a public key denoted K^{-1}
  - Public key known to all, used to encrypt by anyone
  - Private key only known to listener, used to decrypt
- Methods require it is hard (computationally infeasible) to compute K (private) from K^{-1} (public)
  - Computational complexity another interesting talk!
- Public key used in S/MIME, GPG, PGP, Internet key exchange, SSL, TSL, SSH, Bitcoin, others

RSA

- RSA = Rivest, Shamir, Adelman, 1977
  - Currently most commonly used public key system
  - Based on: **multiplication fast**, **factoring slow**
- Key generation
  - Generate 2 primes
  - ~1000 bits each
- Encryption
  - Done by anyone with the public key
- Decryption
  - Only done with knowledge of private key
- Requires large key sizes (2048-4096 bits common)

- Various attacks require care in using
  - 300 bit number factored in 20 minutes on i7
  - 1999 factoring RSA-512 took 8400 MIPS years over 7 months
  - 2009 a RSA-512 key factored in 73 days by PC
  - 2010 a RSA-768 key factored (1500 CPU years)
  - Inserting proper random padding into the message
  - Cannot pick “bad” primes else key weak
  - Timing attacks
  - Adaptive chosen plaintext attacks (requires better message padding)
  - Side channel: branch prediction exploited to attack RSA

Elliptic-curve cryptography



- Another public/private key method
- Smaller keys than RSA, much faster processing
- Based on “adding” points on an elliptic curve, which has form $y^2 = x^3 + ax + b$

- Curve25519 – a specially selected, strong curve
  - $y^2 = x^3 + 48666x^2 + x$, over quadratic extension on field 2^255−19
  - 128 bits of security
  - one of the fastest ECC curves
  - not covered by any known patents
  - NIST approved Curve25519 and Curve448 for use by Federal Government

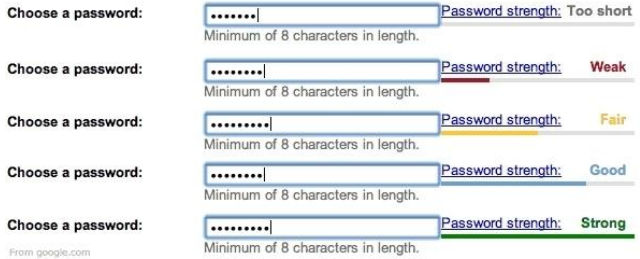
- The elliptic curve Diffie–Hellman (ECDH) key agreement scheme is based on the Diffie–Hellman scheme
- The Elliptic Curve Digital Signature Algorithm (ECDSA) is based on the Digital Signature Algorithm

Guidelines II: Asymmetric Encryption

- Use:
  - ECC
  - NaCl/libsodium
- Avoid:
  - RSA
    - Attacks progressing faster than on ECC
    - ECC much faster
  - ElGamal
  - OpenPGP, OpenSSL, BouncyCastle, etc.
  - Do not implement your own – full of pitfalls

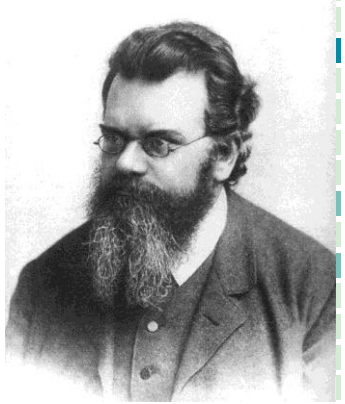
Key length for symmetric algorithms

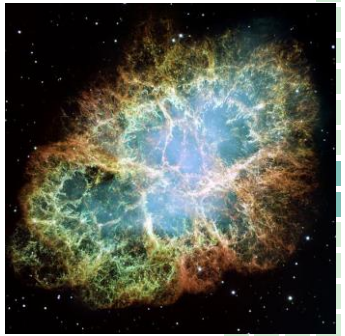
- Usually measured in bits, denoted *n*
- Gives upper bound on algorithm security
- Attacks often weaken security to lower number of bits
- 128 considered decent (until quantum computers common)
- NSA requires 256 bits AES for Top Secret classified data
- 2015: NIST disallows key strength of 112 bits or lower



Thermodynamics limits

- Landauer's Principle : To set or clear a bit requires no less than kT ln(2)
  - k is the Boltzman constant ($1.38*10^{-23}$ J/ºK)
  - T is the absolute temperature of the system in degrees Kelvin
  - ln(2) is natural log of 2, ~0.69
  - (Aside – can do reversible computation with zero energy!)
- Assuming T = 2.73 degrees K (ambient temperature of universe)
  - kT ln(2) = $2.6*10^{-23}$ J per bit change
  - 1J = amount of energy needed to lift average tomato up 1 meter
- Annual energy generation on earth $6*10^{20}$ J = cycle a 144 bit counter
- Annual energy output of the sun $1.2*10^{34}$ J = cycle a 188 bit counter
- Supernova produces in the neighborhood of 10^{44} J = cycle a 221-bit counter





Key length revisited

- Key size for public systems are not comparable with key sizes for private systems
  - Public systems usually have more mathematical “structure”, making them weaker per bit of key size

Symmetric (AES, DES, etc.)	80	128	256
RSA	1024	3072	15,360
ECC	160	256	512

  - NTRU hard to compare, very strong, between Symmetric and ECC
- Larger key sizes require more storage and transmission space, usually result in more computation to use

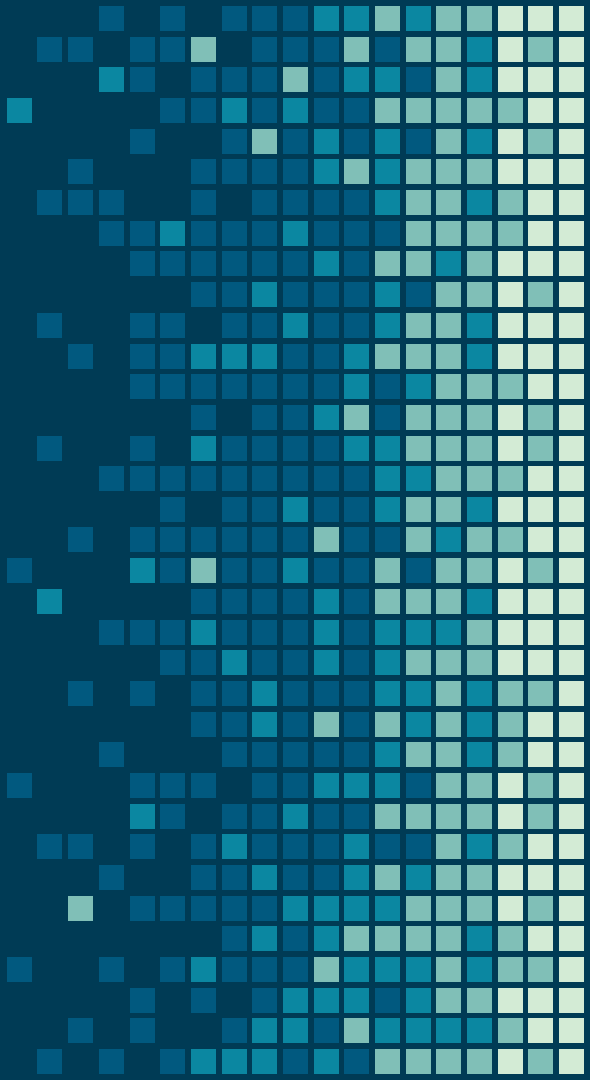
Guidelines III: Key lengths

- Symmetric algorithm (AES, …) key lengths
  - Use:
    - 128 bit minimum key length, preferred 256
    - AES key is far less likely to be broken than your public key pair
  - Avoid:
    - Constructions with huge keys (slow, usually nonsense)
    - Cipher "cascades"
    - Key sizes under 128 bits
- Public key lengths (RSA, ECC, …)
  - NOTE public keys usually much larger than symmetric keys
  - Use:
    - 256-bit minimum for ECC/DH Keys , preferred 512 bit
    - 2048-bit minimum for RSA/DH Group, preferred 4096
  - Avoid:
    - Anything smaller

Passwords

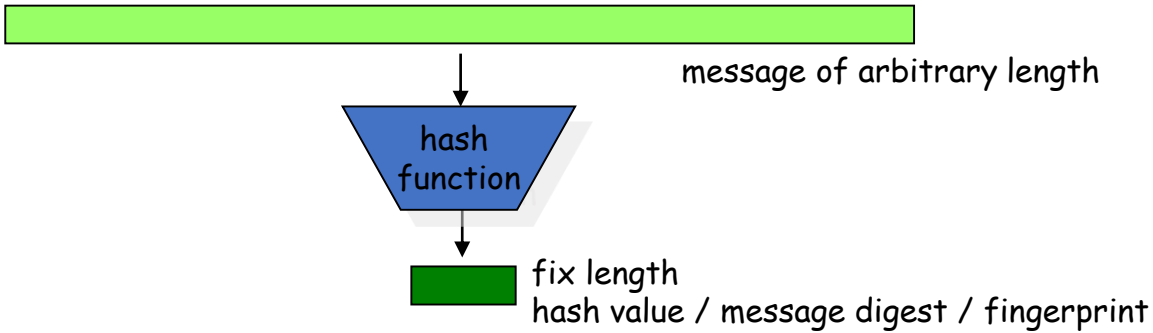
Break all the things!

“Cryptography is typically bypassed, not penetrated” -- Adi Shamir



Hashing

- Hashing takes arbitrary length bit-string into fixed length n bits
- Useful to protect files from tampering
- Collisions unavoidable
- A secure hash makes finding collisions hard
- Message m is hashed by hash function h as h(m)

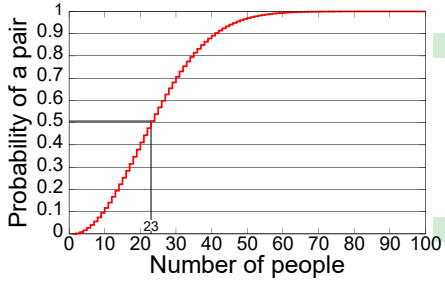


Hashing

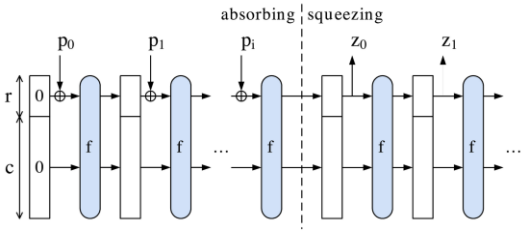
- Desires:
  - Efficient: given m, h(m) easy to compute
  - Weak collision resistance: given m, hard to find m' so h(m)=h(m')
  - Strong collision resistance: hard to find m, m' so h(m)=h(m')
  - One-way: given hash value y, hard to find x so that h(x)=y

- Avalanche effect: changing a single bit of input changes each output bit with 50% probability (SHA-3)
  - `SHAKE128("The quick brown fox jumps over the lazy dog", 256)`
    `f4202e3c5852f9182a0430fd8144f0a74b95e7417ecae17db0f8cfeed0e3e66e`
  - `SHAKE128("The quick brown fox jumps over the lazy dof", 256)`
    `853f4538be0db9621a6cea659a06c1107b1f83f02b13d18297bd39d7411cf10c`

- Birthday paradox
  - Likely collision in few items
  - For n-bit hash, if there are $2^{n/2}$ hashes, there is a likely collision
  - 160-bit or higher recommended for most applications



SHA-3



- Bertoni, Daeman (AES), Peeters, Assche
- Novel “sponge” construction
  - Data “absorbed” into subset of state via XOR
  - State transformed via efficient permutation function f
  - Hash “squeezed” out from same state subset, blocks alternate with f
  - Extra internal state prevents *length extension attack*
    - Lets hacker compute H(message1 + message2) given H(message1)
    - Attack possible on SHA-2, SHA-1, MD5, others
- Internal state 1600 bits (5x5x64), 24 rounds, Padding is 10*1
- Defines two arbitrary output length streams (stream ciphers)
  - SHAKE128 and SHAKE256
- Fast in hardware, slow in software (2x over SHA-2, 3x over SHA-1)

SHA-3

- SHA-3 sizes and bit security, cycles per byte on Intel Skylake

	SHA3-224	SHA3-256	SHA3-384	SHA3-512	SHAKE128	SHAKE256
Output bits	224	256	384	512	d (any)	d (any)
Block size	1152	1088	832	576	1344	1088
security	112	128	192	256	Min(d/2,128)	Min(d/2,256)
Qc security	74	85	128	170	Min(d/3,128)	Min(d/3,128)
Skylake cpb	8.12	8.59	11.06	15.88	7.08	8.59

Guidelines IV: Hashing / HMAC

- Use:
  - Hash with truncated output (sidesteps length extension attacks)
  - SHA3-512
  - SHA3-256
  - HMAC-SHA-512/256

- Avoid:
  - HMAC-MD5,HMAC-SHA1, MD6
  - Custom "keyed hash" constructions
  - Complex polynomial MACs
  - Encrypted hashes
  - Anything CRC

Password background I

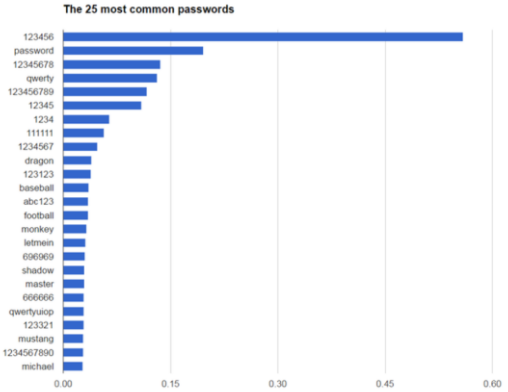
THIS IS NOT THE FINAL ANSWER – DO NOT DO THIS!

- Computer systems requiring password authentication should not store a list of passwords (same for credit cards, etc.)
  - Such a list, if stolen, allows access, making it a target

- First idea: hash the password, store the hash
  - To authenticate user, get password, hash it, compare
  - Stops attacker from using stolen (hash of) password file, since she cannot enter the password.

- How to break this?

Password background II



- Users choose poorly
  - many passwords common, not much entropy
- Attacker steals list of hashed passwords
  - Assumes weak passwords exist
  - Hashes most frequent passwords
  - Can find matches, use to access
- Making list only needs done once, can be used to attack many places efficiently
- So how about storing 37th hash? Or 101st hash? Now attacker needs new tables
  - Attacker tables grow unwieldy
- Rainbow tables
  - Rainbow table: for each common password, hash N times, store password and last hash.
  - Give unknown hashed password, hash it N times, check table each time.
  - If a hit, then the password is most likely the one starting the chain.
  - Breaks places that store hashed passwords as long as table computed to higher N than stolen password
  - Many other flavors of rainbow tables used to attack different things

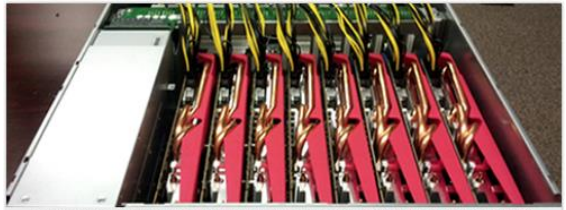
Password background III

- Rainbow table protection
  - Mix in some randomness so table is useless
  - Random “salt” is added: store salt and hash(concat(salt,password))
  - Authentication then gets user password, looks up salt, hashes, checks

- Salt
  - **Must be generated in cryptographically secure, random manner**
  - **Must never be reused**
    - Each password and password change generates a new one

Password background IV

- Still possible to brute force a password
  - Brute force (ever had to crack a zipfile?)



Inside a recent GPU-based hash- cracker - more than 100 billion hashes/sec for $20,000

- Given stolen, salted, hashed password list
  - For each possible or common password, run algorithm on it, check list
  - Can be massively parallelized (CUDA, AWS, FPGA, ASIC, etc.)

- Make this more expensive!
  - Iterations: more iterations of slow hash functions forces attacker to use more time to check
  - Entropy: larger space of passwords makes attacker search longer

Password background V

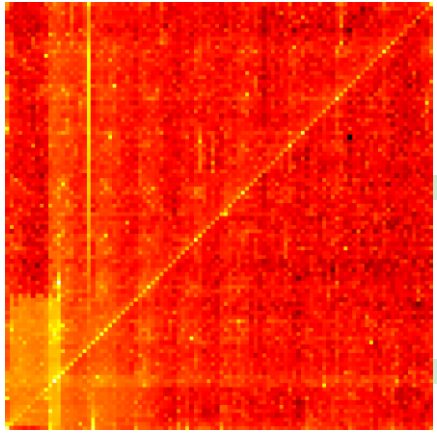
- Entropy – size of password space
  - More letters: require upper+lower
  - Add numbers, symbols
  - Make longer
- More efficient to simply use longer passwords
  - 10 char upper+lower is weaker than 13 long lowercase only $\mathbf{52^{10} < 26^{13}}$
  - 10 char upper+lower+digits+symbols is weaker than 14 long lowercase only $\mathbf{72^{10} < 26^{14}}$
- Doubling iteration count (cheap) doubles computational power (expensive) needed for brute force

Password background VI

- Many good libraries/protocols for this
  - All use iteration count and above knowledge
  - **PBKDF2**
    - RSA designed, still recommended
    - Takes any hash function
    - Small hardware footprint makes it cheaper to attack with hardware
  - **bcrypt** (1999)
    - Blowfish based
    - Slower than PBKD, uses lots of RAM to make ASIC attacks harder
  - **scrypt** (2009, RFC 7914) – newer
    - Newer = riskier, slower
    - Uses exponential time and RAM to push back hardware attacks
    - Most hardware attack resistant
- Over time, increase iteration count to make stronger
  - Simply store iteration count with salt and hash

Password/passphrase evidence

- 2012, Bonneau & Shutova
  - Attacked passphrase dataset that was leaked.
  - Found ~8,000 phrases using 20,000 phrase dictionary (1% of passphrases)
  - Computed that this is 20 bits security for attacking 1% of accounts
  - Standard passwords provide 10 bits against 1% of account attacks
  - **Need to rate limit password/passphrase attempts!**
- PIN heatmap
  - 3.4 million PINs
  - Top 20 most common are ~20% of all
    - 1234 = 10%, 1111 = 6%, 0000 = 1.8%, 1212 = 1.2%
  - What are the bright specks/lines/blocks?



Password enlightenment I

In summary:

1. **Salt**: never reuse, new for each password and password change, length >= hash block length
2. **Hash**: MD5 bad, SHA-1 bad, SHA-2 usable, SHA-3 good. (MD5, SHA-1, SHA-2 allows length extension attacks)
3. **Iterations**: 50K or more, check current recommendations when implementing
4. **Library**: scrypt, bcrypt, or PBKDF2
5. **Store**: iteration count, salt, and hash (and maybe hash algorithm, allowing future changes) in database.
6. **Increase**: iteration count over time as hardware speeds improve
7. (Optional)
   - **Key**: add extra key used in hash, hide that key, helps protect
   - **Erase**: don't keep things around in memory you don't need, never keep passwords in memory longer than needed, or on swap files, or anywhere!
   - **Multi-factor**: use multi-factor authentication when possible

Password enlightenment II

- Good password hygiene
  - Large entropy: longer is better than more variety, 14+ chars pretty good, 20+ excellent
  - Keep weird – don't use common phrases
  - Avoid predictability – if required to use special chars, mix them, not simply at end or beginning
  - Never reuse passwords – use a password manager to manage many
- NIST released new guidelines (Jun 22, 2017, SP 800-63-3)
  - **Remove periodic password change requirements** – multiple studies show this to be counterproductive
  - **Drop required password character complexity**: no more required upper/lower + digits + symbols. This results in worse passwords.
  - **Require screening** of new passwords against common words and phrases
- As result – best practices is now long, unique passphrases, no needed rotation, no special characters required.
  - Maybe a sentence, not a common or cultural phrase.

Guidelines V: Passwords

- Use, in order of preference:
  - scrypt
  - bcrypt
  - sha512crypt
  - sha256crypt
  - PBKDF2
- Avoid:
  - Plaintext
  - Naked SHA-2, SHA-1, MD5
  - Complex homebrew algorithms
  - Any encryption algorithm

Other needed protocols

Secure all the things!

“If you think cryptography is going to solve your problem, you don't understand cryptography and you don't understand your problem.” -- Roger Needham

Speed

- Private key much faster than public key

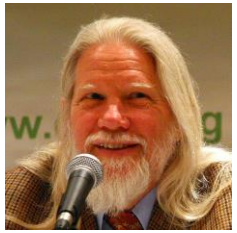
- From Crypto++ library timings
  - Fedora release 25 (x86_64)
  - 6th gen Skylake CPU, 3.1GHZ

- If you invent a secure public key method as fast as private, you'll be a billionaire

Algorithm	MiB/s
AES-CBC (256 bit)	806
MD5	493
ChaCha20 (256 bit)	499
DES-CTR (64 bit)	78
RSA 2048 Encryption	7.8
RSA 2048 Decryption	0.23
ECC (ECIES) 256	0.22
NTRU	~40

Key exchange

- To be a useful encryption method, must have a good way for users to agree on keys
- Diffie-Hellman (1976) published the Diffie-Hellman key exchange
  - Based on fact $(g^a)^b = (g^b)^a = g^{ab}$
    - Computed mod p
    - Everyone knows p and g, including attackers
    - a = Alice secret choice
    - b = Bob secret choice
    - After exchange, Alice and Bob agree on value of $k=g^{ab}$ as secret key
    - Based on fact raising to power is computationally easy but taking root is computationally hard
    - Only transmitted items are g^a and g^b, from which attackers cannot easily compute a or b
  - Doesn't do authentication: man in the middle possible
- Often called Diffie-Hellman-Merkle to recognize Merkle's contribution



Guidelines VI: Key Exchange (Diffie Hellman)

- Use, in order of preference:
  - NaCl/libsodium
  - 2048-bit Diffie-Hellman Group #14
  - Curve25519 – heavily analyzed and tested

- Avoid:
  - conventional DH
  - SRP
  - J-PAKE
  - Handshakes and negotiation
  - Elaborate key negotiation schemes that only use block ciphers
  - srand(time())
  - Implementing your own – very hard to get right

Random number generators (RNGs)

- Random: cannot be predicted by an observer before generated.
- Pseudo-random number generator (PRNG) creates "random" looking numbers from deterministic algorithm
  - PRNGs are entire field
  - "Anyone who attempts to generate **random** numbers by deterministic means is, of course, living in a state of sin." - John von Neumann

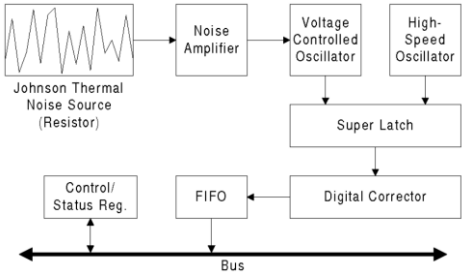
- Crypto needs lots of them
  - Key generation, one-time-pads, salts (later in talk), initialization vectors, more
- ***Cryptographically secure*** pseudo-random number generator (CSPRNG)
  - Passes statistical randomness tests
  - Next-bit security: given k bits, no polynomial time algorithm gives k+1 bit
  - State compromise resistance: knowing some state doesn't give more state
  - Hold up under other attack as needed
  - Do not use normal PRNG when CSPRNG needed!

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

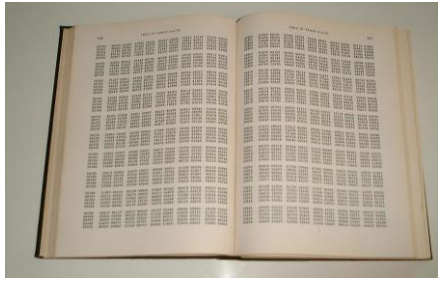
Obligatory XKCD

Random number generators (RNGs)

- Steps:
  - Get entropy from environment
    - Network, keyboard, mouse, electrical noise, radioactive decay, system state
  - Apply proper mixing to remove bias (called whitening)
- Many CPUs now support in hardware
  - Intel has RDRAND instruction taking 117 cycles for 64 bit value
  - AMD has similar, takes ~2500 cycles
  - Theodore T'so resisted using RDRAND in Linux /dev/random after NSA caught trying to weaken encryption
- ***"A Million Random Digits with 100,000 Normal Deviates"*** (1955)
  - A 400 page book with 10^6 random digits produced by a roulette wheel
  - Many humorous reviews on Amazon
- Random.org – online generation from weather noise.



Intel RNG

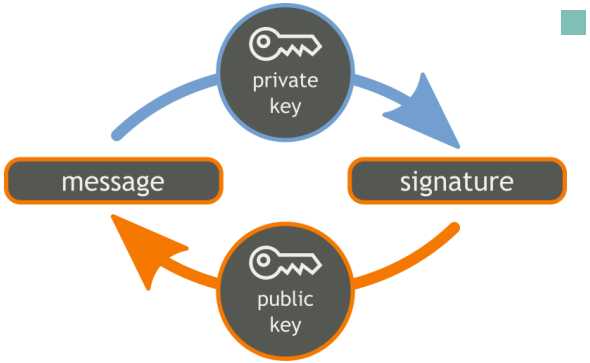


Guidelines VII: Random numbers

- Use:
  - Cryptographically secure numbers
  - /dev/urandom
  - CryptGenRandom
  - Create: 256-bit random numbers

- Avoid:
  - Userspace random number generators
  - /dev/random
  - Anything vanilla C/C++

Digital signatures



- Used for message authentication and non-repudiation of origin
  - Can prove (legally) that someone signed a document digitally
- Based on public key crypto
  - Sign with private E(m,k)=s
  - Verify with public V(m,k') = true if E(m,k)=s, else false
- Slow public key methods:
  - Hash message with fast hash, sign the hash with slow public key method
  - Others cannot fake the signature, but can verify it
- Methods:
  - RSA
  - Digital Signature Algorithm (DSA) (ElGamal based)
  - ECDSA (Elliptic Curve DSA) (elliptic curve based, 320+bits)

Guidelines VIII: Digital signatures

- Use, in order of preference:
  - NaCl/libsodium (as usual!)
  - Ed25519
  - RFC6979 (deterministic DSA/ECDSA)
- Avoid:
  - RSA-PKCS1v15
  - RSASSA-PSS with MGF1+SHA256
  - Really, anything RSA
  - Vanilla ECDSA
  - Vanilla DSA

Filesystems

- Full Disk Encryption (FDE)
  - Most PC drives support
  - SSDs vs platter
  - Software: MBR (often) unencrypted
    - Ex: VeraCrypt (TrueCrypt fork)
  - Hardware: all encrypted

- FileVault (2003+, Mac)
  - AES-CBC
  - FileVault2 AES-XTS

- BitLocker (2004+, Windows)
  - AES-CBC, 128 or 256 bit key
  - Win10 adds FIPS compliant AES-XTS

- iOS
  - Was considered very secure
  - Elcomsoft claims iOS 11 introduces many weaknesses
  - 11.1 broken one day after release

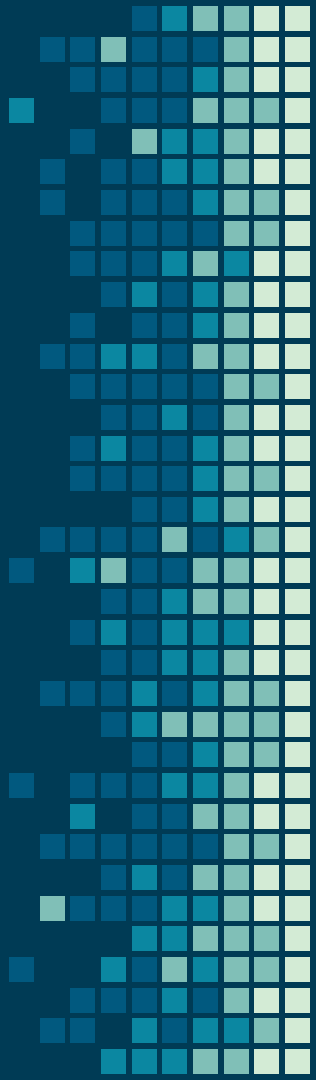
- Android
  - Mandatory in Marshmallow 6.0+ if hardware allows
    - AES-XTS, AES-CBC-ESSIV
  - FDE on some earlier phone models

THANKS!

Any questions?

You can find me at:
clomont@logikos.com
Chris@Lomont.org

"If you think cryptography is going to solve your problem, you don't understand cryptography and you don't understand your problem." -- Roger Needham



Sources

- AppA-Crypto
  - Appendix A: Introduction to cryptographic algorithms and protocols, 2007 Levente Buttyán and Jean-Pierre Hubaux, http://secowinet.epfl.ch/, Security and Cooperation in Wireless Networks
- Computer Science Illuminated, 4th ED, Chapter 12, Erin Chambers edited slides
- Crypto benchmarks http://bench.cr.yp.to/
- Wikipedia
- TODO more
- Standards agencies
  - FIPS, ANSI, ISO, IEEE, IETF, NSA, GCHQ

Modern Cryptography

Science replaces the invent-break-tweak loop

“One must acknowledge with cryptography no amount of violence will ever solve a math problem.” -- Jacob Appelbaum, “Cypherpunks: Freedom and the Future of the Internet”

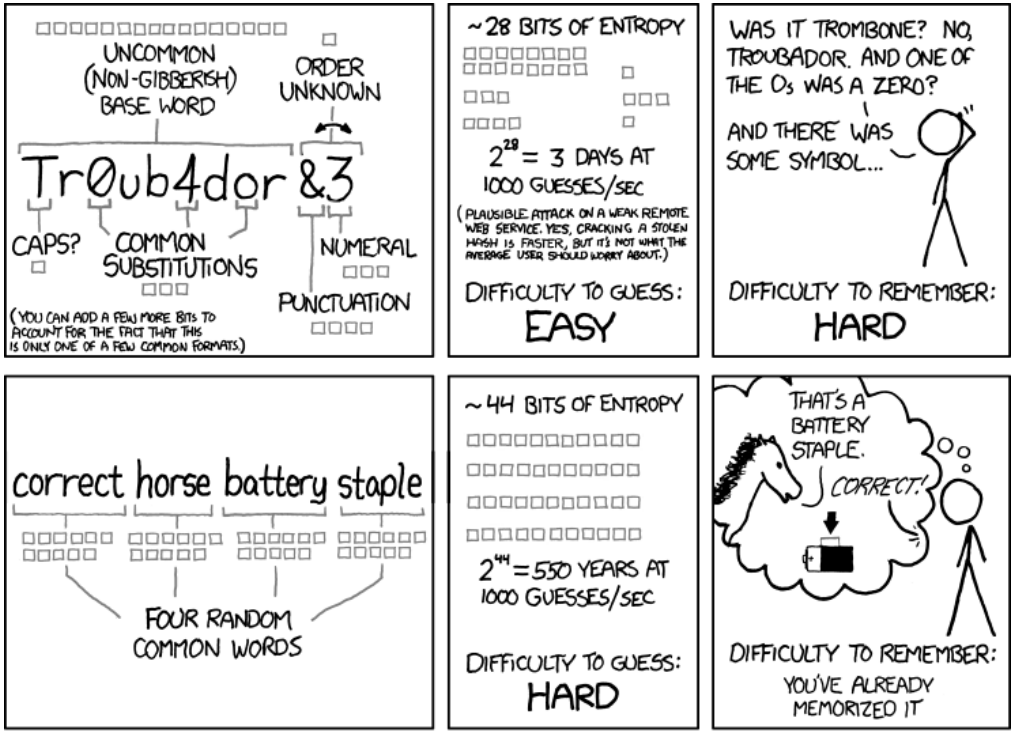
Two Factor Authentication

- TODO – why?
- List attacks
- Banks do it ☺
- There are three types of authentication:
- Something you know: a password, PIN, zip code or answer to a question (mother's maiden name, name of pet, and so on)
- Something you have: a phone, credit card or fob
- Something you are: a biometric such as a fingerprint, retina, face or voice

Hashing

- Lots of hash functions
  - SHA, MD5, Cubehash, Tiger, Whirlpool, hundreds more
- Like PRNGs, some hash functions not secure.
  - Use a cryptographic hash function
- Over time many, many weak or broken
  - 2004: SHA-0, RIPEMD, MD5 shown weak
  - 2005: SHA-1 reduced in strength from 2^80 to 2^59
  - 2009: most popular MD5 and SHA-1
  - 2008: successful MD5 attack breaks TLS
  - 2017: Google announces SHA-1 collision
- NIST held new has competition starting in 2006
  - 2012: Winner called SHA-3
  - SHA-3 and BLAKE2 both considered secure today
  - SHA-2 still ok, but on way out

Password enlightenment XKCD



Advanced Topics

More math, more quantum

“How long do you want these messages to remain secret?[...]
I want them to remain secret for as long as men are capable of evil.”
— Neal Stephenson, “Cryptonomicon”

- Website security
- By "website security", we mean "the library you use to make your web server speak HTTPS". Believe it or not, OpenSSL is still probably the right decision here, if you can't just delegate this to Amazon and use HTTPS elastic load balancers, which makes this their problem not yours.
- Use:
  - OpenSSL, LibreSSL, or BoringSSL if you run your own site
  - Amazon AWS Elastic Load Balancing if Amazon does
- Avoid:
  - PolarSSL
  - GnuTLS
  - MatrixSSL

- Client-server application security
  - What happens when you design your own custom RSA protocol is that 1-18 months afterwards, hopefully sooner but often later, you discover that you made a mistake and your protocol had virtually no security. A good example is Salt Stack. Salt managed to deploy e=1 RSA.
  - It seems a little crazy to recommend TLS given its recent history:
  - The Logjam DH negotiation attack
  - The FREAK export cipher attack
  - The POODLE CBC oracle attack
  - The RC4 fiasco
  - The CRIME compression attack
  - The Lucky13 CBC padding oracle timing attack
  - The BEAST CBC chained IV attack
  - Heartbleed
  - Renegotiation
  - Triple Handshakes
  - Compromised CAs
- Here's why you should still use TLS for your custom transport problem:
  - Many of these attacks only work against browsers, because they rely on the victim accepting and executing attacker-controlled Javascript in order to generate repeated known/chosen plaintexts.
  - Most of these attacks can be mitigated by hardcoding TLS 1.2+, ECDHE and AES-GCM. That sounds tricky, and it is, but it's less tricky than designing your own transport protocol with ECDHE and AES-GCM!
  - In a custom transport scenario, you don't need to depend on CAs: you can self-sign a certificate and ship it with your code, just like Colin suggests you do with RSA keys.
- Use: TLS
- Avoid:
  - Designing your own encrypted transport, which is a genuinely hard engineering problem;
  - Using TLS but in a default configuration, like, with "curl"
  - Using "curl"
  - IPSEC

Other security needs

- Authentication. Verifying validity of a form of identification.
- Integrity checking. Ensures data not tampered with or corrupted.
- Digital Signatures. Electronically sign documents in unforgeable way.
- Non-repudiation. Allows one to prove authorship of some piece of data.
- Zero-knowledge proofs. Alice proves to Bob that she earns <$50K without Bob learning her income.
- Privacy-preserving data mining. Bob holds DB. Alice gets answer to one query, without Bob knowing what she asked.
- Playing poker over the net. Alice, Bob, Carol and David can play poker over the net without trusting each other or any central server.
- Distributed systems. Distribute sensitive data to 7 servers s.t. as long as <3 are broken, no harm to security occurs.
- Electronic auctions. Can run auctions s.t. *no one* (even not seller) learns anything other than winning party and bid.
- Secure voting. Provably strong method to ensure votes count, can check yours, no one else can.
- More

RSA

- RSA = Rivest, Shamir, Adelman, 1977
  - Currently most commonly used public key system
  - Based on: **multiplication fast**, **factoring slow**

- Key generation
  - select p, q large primes (about 500-1000 bits each)
  - Let n = p q, let ϕ = (p-1)(q-1)
  - Select e such that 1 < e < ϕ and gcd(e, ϕ) = 1
  - Compute d such that e d = 1 mod ϕ
    - **this is easy if ϕ is known**
  - Public key is (e, n)
  - Private key is d

- Encryption
  - Done by anyone with the public key
  - Represent the message as an integer m in [0, n-1]
  - Compute ciphertext c = m^e mod n

- Decryption
  - Only done with knowledge of d
  - Compute m = c^d = m^{ed} mod n
  - Last works based on number theory
    - (Euler's generalization of Fermat's Little Theorem)

RSA

- Requires large key sizes (2048-4096 bits common)
  - Currently can factor 300 bit number in around 20 minutes on i7
  - 1999 factoring RSA-512 took 8400 MIPS years over 7 months
  - 2009 a RSA-512 key factored in 73 days by PC
  - 2010 a RSA-768 key factored (1500 CPU years)

$$O\left(\exp\sqrt[3]{\frac{64}{9}b(\log b)^2}\right).$$

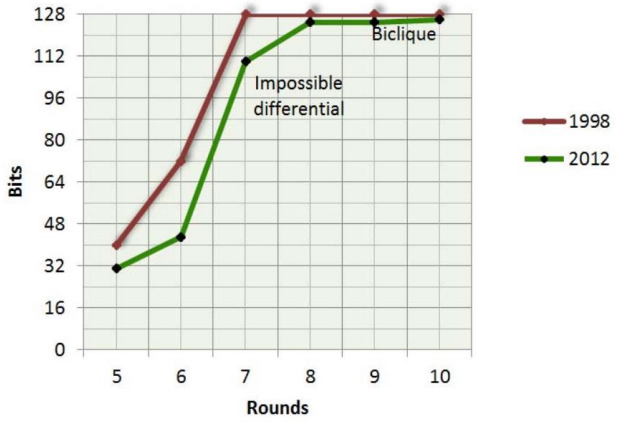
- Various attacks require care in using
  - Inserting proper random padding into the message
  - Cannot pick “bad” primes else key weak
  - Timing attacks
  - Adaptive chosen plaintext attacks (requires better message padding)
  - Side channel: branch prediction exploited to attack RSA

Hard problems

- So the search was on for such public and private key relations
  - (complexity classes P, NP, etc., for another talk!)
- **Diffie-Hellman problem** (1976)
  - given a prime p, a generator g of Z_p^*, and elements $g^x \bmod p$ and $g^y \bmod p$, find $g^{xy} \bmod p$
- **Factoring problem** (RSA, 1978)
  - given a positive integer n, find its prime factors
- **Discrete logarithm problem** (ElGamal, 1985)
  - given a prime p, a generator g of Z_p^*, and an element y in Z_p^*, find the integer x, $0 \leq x \leq p-2$, such that $g^x \bmod p = y$
- For all three, ***the true complexity is unknown***
  - Believed not in P, the class of efficiently solvable problems

Modern AES attacks

- Some *implementations* have side-channel attacks
  - 2005 Bernstein uses a cache-timing attack to break a custom OpenSSL server, required 200M chosen plaintexts
  - 2005 Shamir et. al. user mode gets root mode key in 65ms from timing attacks
- 2009+: many small, non-practical attacks
  - reducing the keyspace by factor of 4, for example
- Many attacks work on the round key generation
  - It's too simple



Rounds	AES-128	AES-192	AES-256
8	125.4		
10	126.2		
12		189.7	
14			254.2

- TODO – abstract: Most developers have heard the statement: “never invent your own cryptography,” and most don’t fall into that trap, yet even using well designed libraries can land you in hot water, through misunderstanding, using them incorrectly, or only using them in a few of the many places you need security. This talk will cover many common errors and how to avoid them, and will increase your knowledge of how to avoid the mistakes that routinely leaks user data. We will cover data at rest, password management and best practices, two factor authentication, and more.

- Data at rest
- Password management
- 2-factor authentication
- Get quotes from old section breaks

Modern crypto basis

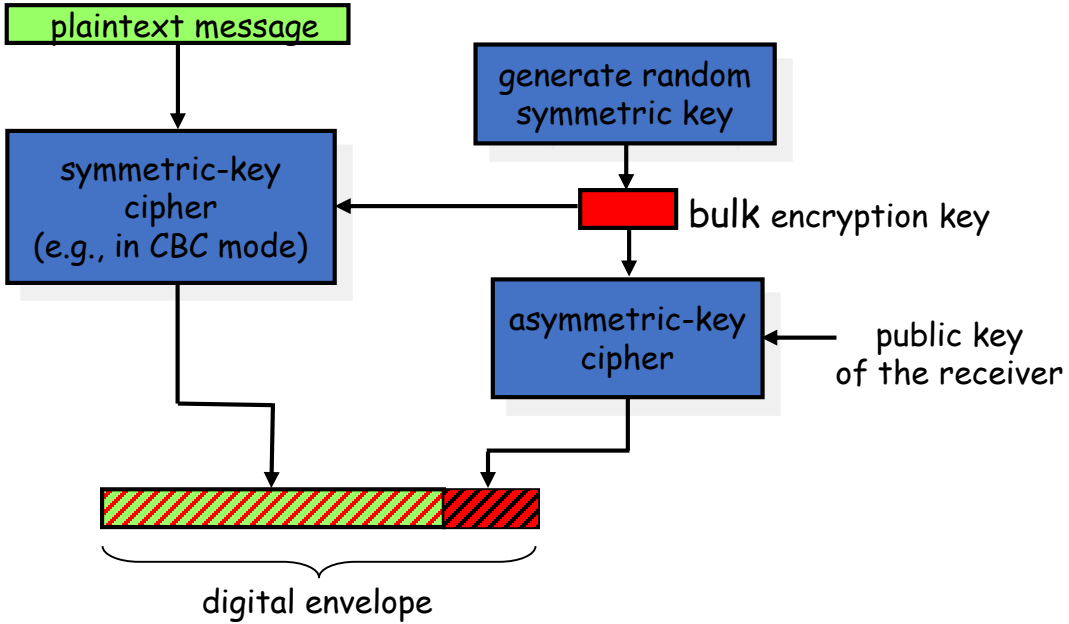
- Kerckhoffs's Principle (1883): A cryptographic system should be secure even if everything about the system, except the key, is public knowledge.
- Shannon's maxim: "**The enemy knows the system**"

Claude Shannon
Started Information Theory
Introduced the word 'bit'

- Provable security breaks "invent-break-tweak" cycle
  - Perfect Security (Shannon)
  - Vast improvement on generating "random" numbers
  - One-way functions thwart massive computational power
- Math and techniques climb to bewildering complexity

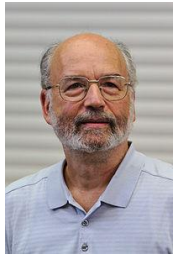
Digital envelope

- Private key much faster than public key
- Public key methods used to encrypt private key, private key used to encrypt, called a digital envelope



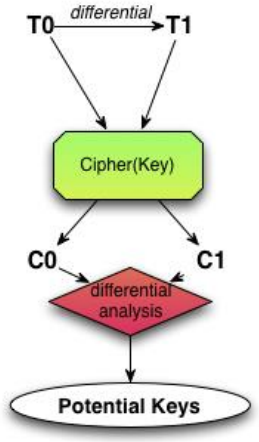
Attacks (Part I – Differential Cryptanalysis)

- Studies how differences in inputs can affect differences of outputs
  - Biham and **Shamir** in late 1980s
    - Used to attacks various ciphers and hashes
  - Later found IBM and NSA knew about it earlier
  - Usually a *chosen plaintext attack*



Adi Shamir

- DES designed (secretly) to be resistant to this (2^55 plaintexts to crack 56 bit key)

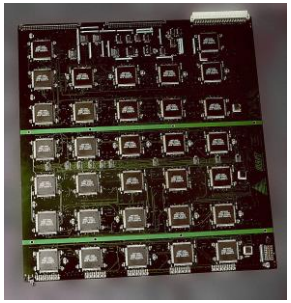


- Did break FEAL, another cipher
  - Original FEAL, 4 rounds, broken with 8 chosen plaintexts
  - 31 round FEAL susceptible

- Used to attack (not break) DES
  - Successful at 2^47 chosen plaintexts

Attacks on DES

- In 1997, broken by Internet in a few months.



Deep Crack had 1856 custom chips

- In Jul 1998, Deep Crack
  - Specialized machine built by the EFF
  - Cost $250K, took 6-8 months to build, cracks message in a few days
- In 1999 above combined to break in 22 hours

- Currently FPGA devices break it regularly in around a day at low (~$10K) cost

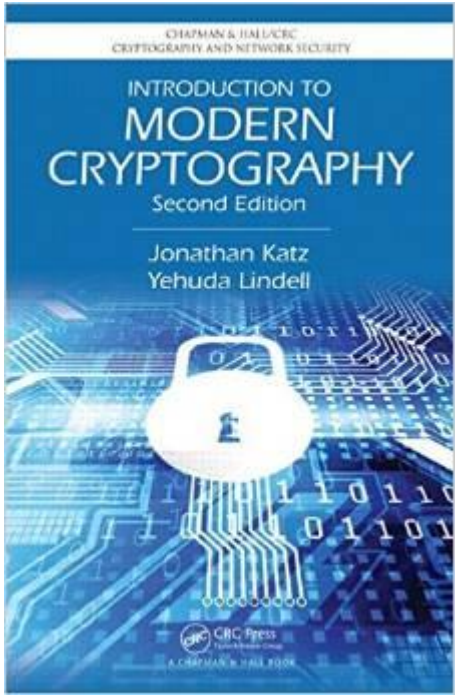
TripleDES

- Salvage DES by repeating DES three times with 3 keys (168 bit key)
  - Result called Triple DES (1998)

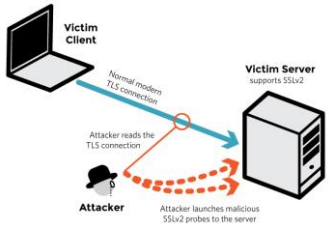
- Cons
  - Key Option 1: due to the meet-in-the-middle attack (later in talk), the effective security it provides is only 112 bits
  - Key Option 2: effective security 80 bits, easily broken today
  - OpenSSL removed 3DES in 2016, calling it “weak”

Modern cipher design

- Need good key management
  - Else key schedule attacks
- Need good data mixing/substitution
  - Else frequency analysis
  - P-box, S-box combinations
- Need ***high throughput*** in software and/or hardware
  - Leads to lots of tradeoffs, analysis
- Leads to the modern **Advanced Encryption Standard**



Attacks and breaks

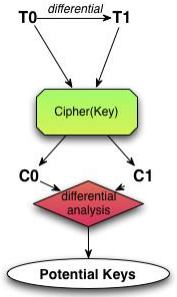


- We'll split into
  - Direct crypto attacks – attacks on the cipher directly
  - Software and hardware attacks – implementation errors or leaks
  - Other side channel attacks
- Direct attack terminology – what attacker has
  - ciphertext only – access only to a set of ciphertexts
  - chosen ciphertext – decryptions of some chosen ciphertexts
  - known plaintext – access to plaintext and ciphertext pairs, attacks key
  - chosen plaintext – ciphertexts for arbitrary (attacker chosen) plaintexts
- Most breaks on modern crypto done not to the main algorithm, but to side issues

Direct attacks

- Brute Force Attack (also see password section)
  - Trying many passwords or passphrases with the hope of eventually guessing correctly
  - Who has tried to brute force an encrypted zip file?
  - Incredibly powerful machines built to do this
  - NSA?
  - Many tools:
    - Aircrack-ng, Cain and Abel, Crack, DaveGrohl, John the Ripper, L0phtCrack
- 1995: Netscape SSL cracked by bad RNG
- 2008: OpenSSL in Debian/Ubuntu flawed

Differential cryptanalysis



- Eli Biham and Adi Shamir 1980s
- Exploit how differences in plaintext relate in ciphertext
- Exploits asymmetry to deduce key
- Primarily for block ciphers, less so for stream ciphers and hash functions
- Don Coppersmith (IDM DES team) published in 1994 that IBM knew of differential in 1974
  - "After discussions with NSA, it was decided that disclosure of the design considerations would reveal the technique of differential cryptanalysis, a powerful technique that could be used against many ciphers. This in turn would weaken the competitive advantage the United States enjoyed over other countries in the field of cryptography."
- Breaks FEAL after 4 rounds with 8 chosen plaintexts, even breaks 31 round FEAL
- DES with ~2^47 plaintexts

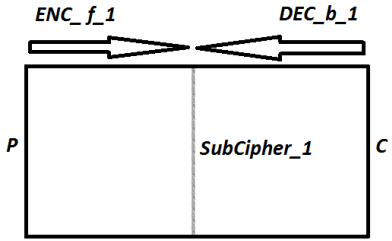
Higher order differential cryptanalysis

- Xuejia Lai, in 1994, laid the groundwork by showing that differentials are a special case of the more general case of higher order derivates

- Lars Knudsen, in the same year, was able to show how the concept of higher order derivatives can be used to mount attacks on block ciphers

- Used to break KN-Cipher, a cipher which had previously been proved to be immune against standard differential cryptanalysis.

Meet-in-the-middle attack



- Tempting to encrypt data several times using multiple keys
- MITM is a generic time-space tradeoff
  - Stores intermediate encryption and/or decryptions values, speeds up brute force
  - Find keys using both the range (ciphertext) and domain (plaintext) of the composition of several functions (or block ciphers) such that the forward mapping through the first functions is the same as the backward mapping (inverse image) through the last functions, meeting in the middle
  - Is why Double DES (116 bits of key) not used - broken with 2^57 ops
  - Is why Triple DES (168 bit) brute forced in 2^56 speed, 2^112 ops
- Multidimensional MITM (MD-MITM)
  - several simultaneous MITM-attacks at once for multiple layers

Related key attack

- Attacker observes operation of cipher operating with different, yet related, keys
- 1994: Biham
- Breaks KASUMI (successor to A5/2)
  - 8 round, 64-bit block, 128-bit key
  - Was basis of 3G confidentiality and integrity algorithms

- Wired Equivalent Privacy (WEP) used in WiFi wireless routers
  - Released 1997,
  - Based on RC4 stream cipher with too weak 24-bit IV should not be used on repeated traffic
  - 24-bit IV implies for 5000 packets, have 50% chance of IV repeating (Birthday Paradox)
  - 2003: Replaced by interim Wi-Fi Protected Access (WPA)
  - 2004: Replaced by more secure WPA2
  - See KRACK in replay attack section

XSL attack

- eXtended Sparse Linearization (XSL) attack is a method of cryptanalysis for block ciphers. The attack was first published in 2002 by researchers Nicolas Courtois and Josef Pieprzyk
- XSL attack relies on first analyzing the internals of a cipher and deriving a system of quadratic simultaneous equations. These systems of equations are typically very large, for example 8,000 equations with 1,600 variables for the 128-bit AES
- Still NP-Hard
- My roommate Jiun-Ming Chen from grad school (we had same advisor T.T.Moh) works on this

- Unlike other forms of cryptanalysis, such as differential and linear cryptanalysis, only one or two known plaintexts are required.

Software attacks

- Timing Analysis – exploit timing differences

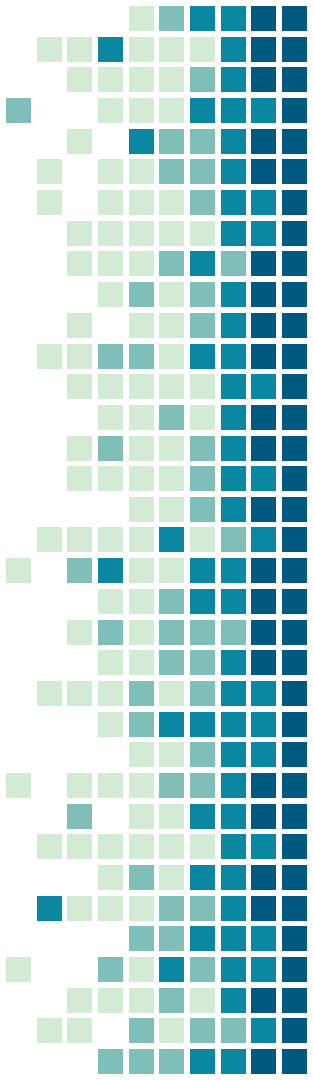
```
if (strcmp(password,input) != 0)
    return false;
```

  - Why is this code bad?
  - Execution time for “square-and-multiply algorithm” used in modular exponentiation linearly on the number of '1' bits in the key
  - 2003: Boneh , Brumley demonstrated practical network-based timing attack on SSL-enabled web servers, based on a different vulnerability having to do with the use of RSA with Chinese remainder theorem optimizations. The actual network distance was small in their experiments, but the attack successfully recovered a server private key in a matter of ***hours***
- Cache leaks
  - monitoring cache timing allows one to attack things in shared environments like VMs, cloud computers, etc.
  - 2005: Bernstein used cache timing for successful AES attacks, 2^27 to recover 128 bit key
- Data remanence leaks
  - items need wiped in RAM and disk (and be careful about swap files, etc.!)
  - SSD page balancing: leaves pages with data if not very careful

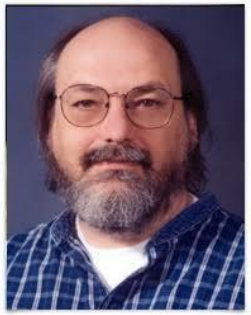
Software attacks

- Implementation failures common
  - Software errors abound: buffer overflows, integer overflows, return oriented programming, etc.

- Nice when happens to malware
  - CryptoDefense - early ransomware, stored key locally
  - Nuclear Exploit Kit – used a key of 0 in Diffie-Hellman, doing nothing
  - Nemucod – claimed RSA1024, used file renaming! (Bluffware?!)
  - Linux.encoder – used srand(time()) instead of srand(md5(time())), last file time allowed decryption



Source code safety?



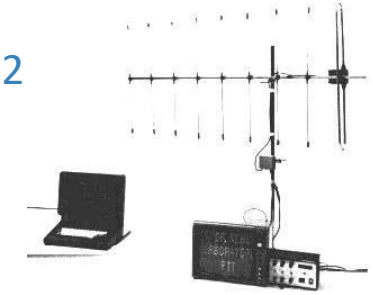
- What if you don't trust a binary?
  - "I'll build it from source"
  - But what about the compiler?
  - Build it from source.
  - But why trust that compiler???

- 1984: Ken Thompson, "**Reflections on Trusting Trust**"
  - Thompson: UNIX, B predecessor to C, UTF-8, ed, Plan 9, Go programming language at Google)
  - Puts a backdoor into a login (or any code) so that there is no source code trace of it!

1. Determine changes needed in compiler source code so it recognizes when it's compiling the target and inserts the backdoor. No code changes made here.
2. Change compiler source to recognize it's compiling itself and inserts the code changes from step 1.
3. Build the compiler with itself.
4. Remove the code from steps 2. There is now no code trace of the hack in either the compiler or target. Compiler will now build poisoned compilers.
5. Now any time the compiler is used to compile itself or the target, the hack continues. ***Neither of them have it in the source code!***

Hardware attacks

- Van Eck Phreaking – electromagnetic emanations leak information
  - 1985: Used on CRTs to read info
  - Works on LCDs too – proof of concept cost around $2
  - 2009: used to read voting ballots
  - Declassified NSA TEMPEST attack

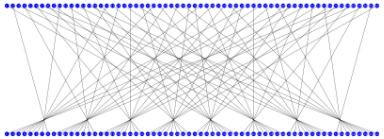


- Power analysis
  - Different operations require different power
  - Simple power analysis (SPA) graphs electrical activity over time.
  - Differential power analysis (DPA) computes intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations.
  - 1998: Paul Kocher, Joshua Jaffe and Benjamin Jun introduce SPA and DPA
  - Used to attack smart cards, extract keys. One example breaks 128 bit ASIC AES in 3 minutes

Symmetric key encryption

- Alice & Bob want to communicate securely
- They agree in advance on 3 components
  - Encryption algorithm: E
  - Decryption algorithm: D
  - Secret key: k
- To encrypt plaintext m, Alice sends c = E(m,k) to Bob
- To decrypt a cyphertext c, Bob computes m' = D(c,k)
- A scheme is valid if m'=m
- Secure if eavesdropper can not learn m from c,
  - *even knowing E and D*

Data Encryption Standard (DES)



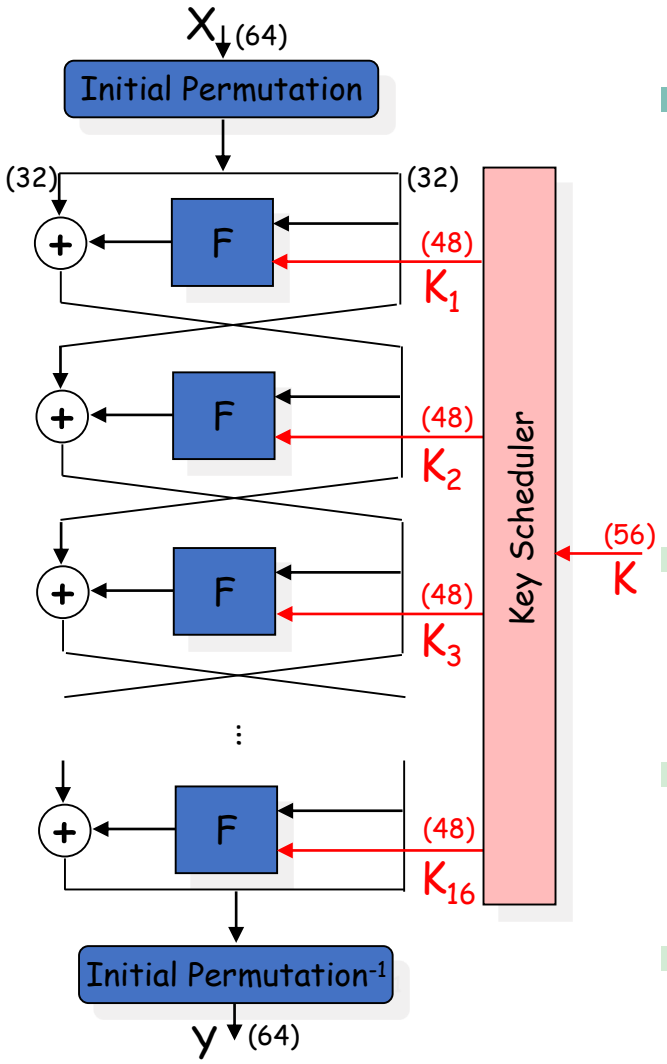
- Well known, early symmetric key encryption standard
- Adopted in 1977 by National Bureau of Standards (now NIST)
- Break message into 64 bit chunks, key size 56 bits
- Main idea: **XOR them together**
- Widespread, e.g., used by HBO for TV scrambling in 1986

DES

	Bits
Input size	64
Output Size	64
Key size	56
Rounds	16

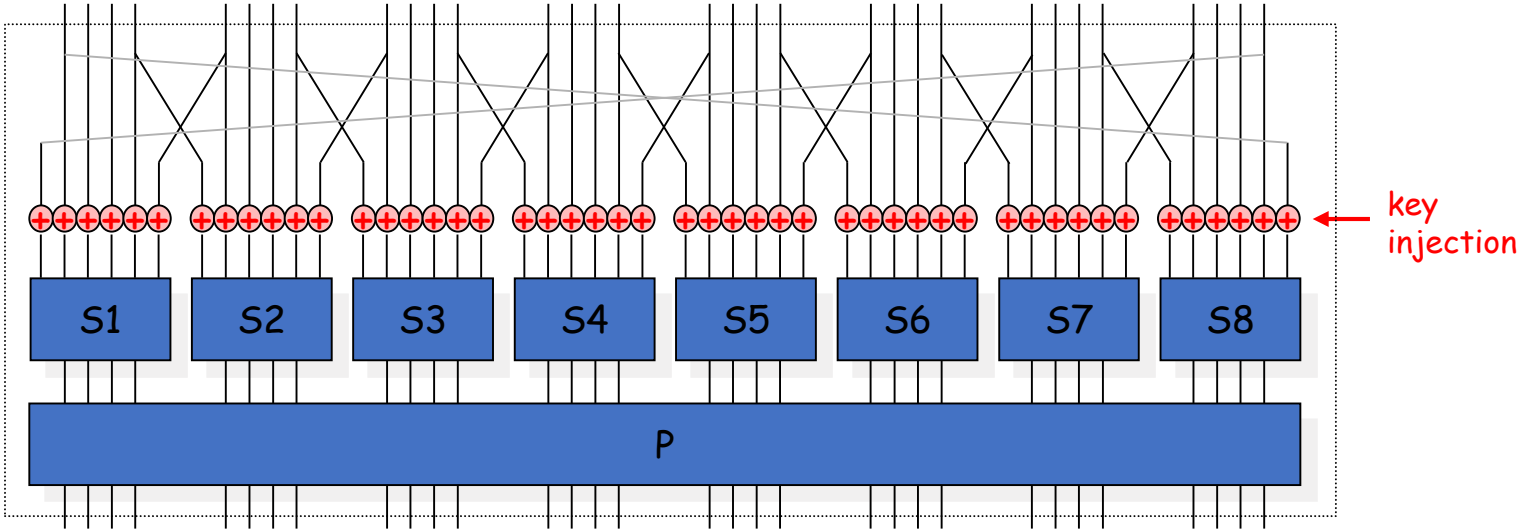
F = Feistel network

Horst Feistel worked at IBM on Lucifer cipher 1970s



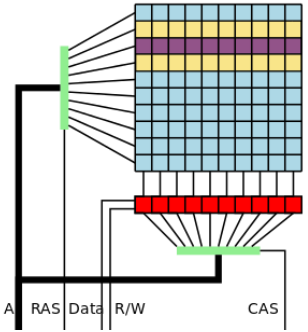
DES

- Substitution-permutation networks introduced by Shannon in 1949
- Si – Substitution box (S-Box)
- P = Permutation box (P-box)

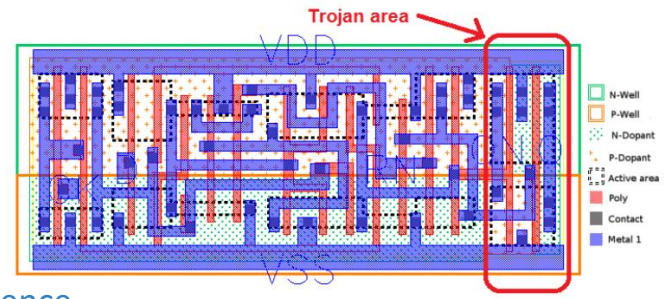


Hardware attacks

- Cold boot: 2008 Ed Felten
  - DRAM/SRAM maintains data without power, slow decay, running computer cold-booted with special OS, copies RAM, gets data
  - Alternatively, machine stolen, frozen in liquid NO2, ram chips pulled and read later
- Rowhammer: 2014
  - Write DRAM repeatedly and quickly: ~400k times in 64ms
  - Can flip bits in neighboring physical memory row
  - 2015: Google demonstrates x86-64 exploits in NaCl and Linux
  - 2015: Proof of concept in JavaScript and Firefox 39



Hardware attacks



- Can have backdoors in hardware such as CPUs
  - DARPA runs program for Trusted Foundries
  - Hardware trojans
    - Could remove NX protection after code sequence
    - Could weaken hardware RNG (see Intel discussion under RGN section)
- 2008: Sam King adds 1400 gates to SPARC, inserts Linux backdoor
- 2014: Kumar, Jovanovic, et. al. Doping level attacks
  - Change chemical composition of gates to modify behavior
  - Needs no gate changed or added
  - Demonstrates weakening of a cipher by fault injection
- Fault attacks
  - Lower voltage to cause hardware hiccups, get info
  - High temperatures to cause hiccups, exploit timing, others
  - Over clocking

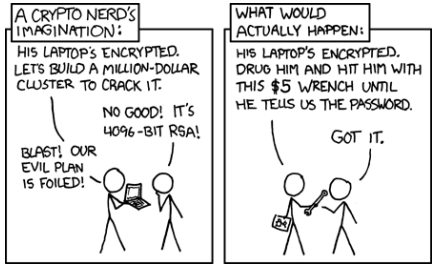
Other side-channel attacks

- Black bag
  - steal keys via burglary, keyloggers, social engineering, credit card skimmer, fake ATM
- Man-in-the-middle (physical version)
  - Authentication and tamper detection used to prevent
  - 2011: security breach at Dutch certificate authority DigiNotar resulted in fraudulent certificates, which were used for MITM attacks
  - 2017: Equifax withdrew mobile phone apps following man-in-the-middle vulnerabilities
  - AT&T – Room 641A, NSA room in building, started 2003, exposed 2006
  - NSA impersonation of Google: document leaked by Edward Snowden diagrams how a "man in the middle attack" involving Google was apparently carried out

Other side-channel attacks

- Replay attack
  - Vehicle key-fob usually hardened against replay attacks, but vulnerable to buffered attacks:
  - RF jammer blocks legit attempts, records message, then when new attempts made, plays old one, storing one ahead for later use.
  - Text-dependent speaker verification
  - Oct 2017: KRACK exploits hole in WPA2 protocol
    - Windows, macOS, iOS, Linux, OpenBSD,….
    - Repeatedly resend handshake to gather info
    - Some vendors offer patch to hide the hole
- Rubber hose – torture to get the key
- Physical insertion of devices
  - USB, FireWire extremely susceptible
  - Safe hacking

$11 car stealing device



Breaks

- Block cipher **Madryga** proposed 1984, not widely used, found susceptible to cipher-only attacks in 1998
- **GOST**, developed in 1970s, published 1989 (Soviet Russian), 256 bit key size, 64 bit block, 2007-2012 breaks in 2^178
- **FEAL-4** proposed as DES replacement in 1987, destroyed by range of practical attacks by 1990.
- **DES** - broken 1990 Biham+Shamir, 2^47, 1998 brute force cracked single-DES (EFF DES Cracker),
- Old 40-bit “export strength” crypto broken in make cases
- DVD Content Scrambling System (**CSS**) 1996 released, cracked 1999
- **SHA-0** - made 1993, revised immediately due to flaw to SHA-1, 1998 attack, 2008 boomerang collisions down to 2^33
- **SHA-1** - made 1995, weakened from 2005 (2^69) to 2017 (2^63)
- **A5/1** (made in 1987, broken 2000), **A5/2**, **CMEA**, **DECT** used in phones, broken in hours, minutes, or seconds
- Stream cipher **RC4** (designed 1987, broken in stages 1994-2013) used in SSL, TSL, WEP, WPA.
- **WEP** broken because of weaknesses in RC4 and poor design allowing related-key attacks, replaced by WPA then WPA2
- **MD5** - designed 1991, collision 5 years later, 2009 reduced to 2^39 (from 2^128)
- 2008, MD5 weaknesses and poor certificate issuer practices allowed collision attacks on hash functions. Issuer fixed practices

Advanced Encryption Standard (AES)

- NIST sponsored competition, started 1997
- Needed to replace 3DES
- Finalists: MARS, RC6, Rijndael, Serpent, Twofish
- AES wins (Vincent Rijmen, Joan Daeman)
  - subset of Rijndael
  - AES has fewer block and key sizes
  - Federal Standard in 2002
  - Only publicly available cipher allowed for Top Secret data
- Modern
- Strong, fast
- Now has lots of hardware support

AES

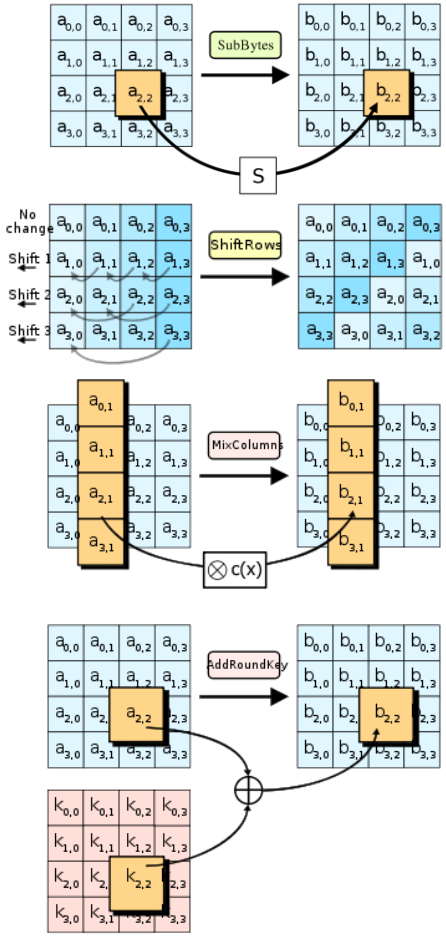
- Key is 128, 192, or 256 bits
- Corresponds to 10, 12, or 14 rounds of cipher
- Bytes treated as entries in *finite field*
  - Addition is XOR
  - Byte 0b1001_0111 is polynomial $x^7 + x^4 + x^2 + x + 1$
  - Multiplication as polynomial multiplication taken as remainder of $x^8 + x^4 + x^3 + x + 1$
- AES S-Box
  - Invert byte in AES finite field GF256, then ->

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

- Designed to resist linear and differential attacks

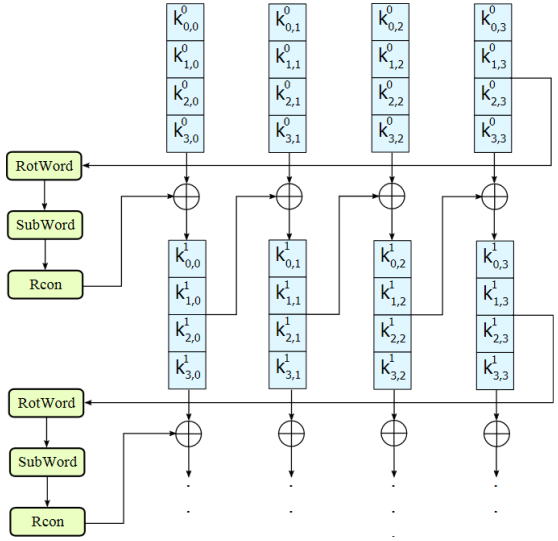
AES

- Stage 1: Substitute
  - Invert byte in GF2^8
  - Apply S-box
- Stage 2: Shift rows
  - Each row cyclically shifted 0,1,2,3
- Stage 3: Mix columns
  - Each column treated as polynomial
  - Multiplied by fixed polynomial
  - Result taken mod x^4+1
- Stage 4: XOR round key
  - Round key derived from main key
  - Uses pieces of rest of algorithm for simplicity



AES key schedule

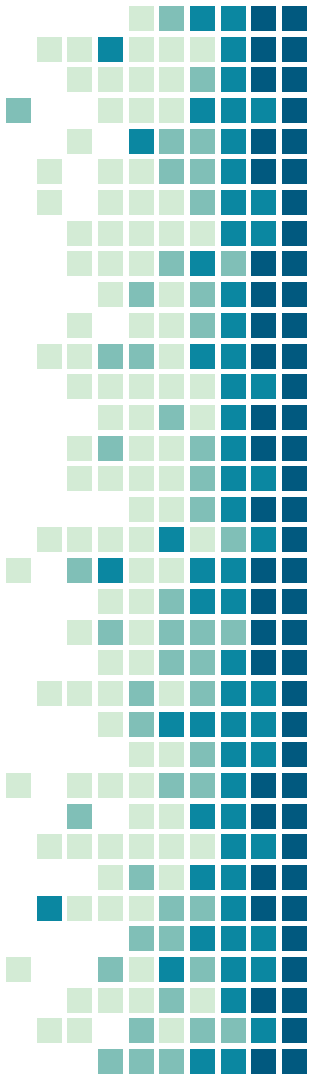
- RotWord
  - Rotates 32 bit word 8 bits to left
- SubWord
  - Apply S-box to each of 4 bytes
- Rcon
  - On leftmost byte, compute $2^{\text{round \#}}$, treat as polynomial in GF256 as before



128 bit key example

Modern AES attacks

- Some *implementations* have side-channel attacks
  - 2005 Bernstein uses a cache-timing attack to break a custom OpenSSL server, required 200M chosen plaintexts
  - 2005 Shamir et. al. user mode gets root mode key in 65ms from timing attacks
- 2009+: many small, non-practical attacks
  - reducing the keyspace by factor of 4, for example
- Many attacks work on the round key generation – too simple



Progress TODO

Basics (9 slides)

- **simple ciphers**
- **frequency analysis**
- **making ciphers**
- **examples**
- **unbroken**

Modern Crypto (AES, RSA, key length, RNGs…) (40 slides)

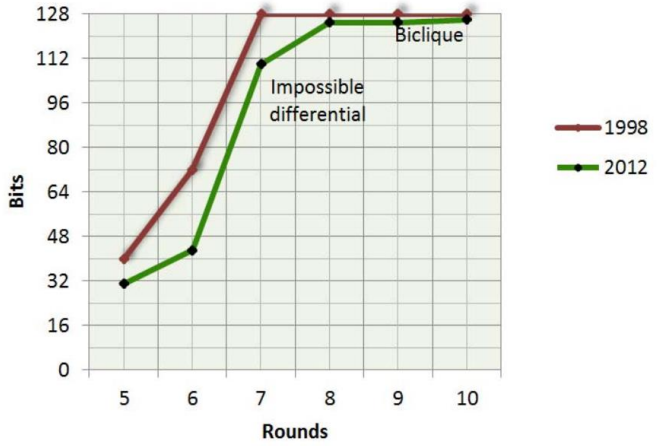
Other Uses (Hashes, Passwords, Stego, Law…) (25 slides)

Attacks and Breaks (Math, Software, Hardware, Famous) (25 slides)

Advanced Topics (Zero knowledge, Quantum, Post quantum) (5 slides)

Modern AES attacks

- Bits of security



Rounds	AES-128	AES-192	AES-256
8	125.4		
10	126.2		
12		189.7	
14			254.2

Terminology

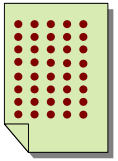
- Alice & Bob want to communicate securely

eavesdropping adversary

m
plaintext

E

K
encryption key



c = E(m,k)
ciphertext

D

K'
decryption key

D(c,K') = m