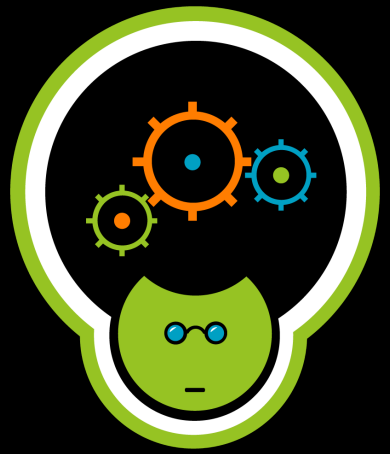
PKI Unlocked: A No-Math Primer for Builders

Jake Hildreth
jakehildreth.com

Code Mash 2026-01-15



semperis

`Get-ADUser -Identity 'Jake Hildreth'`

- Husband, Dad, Recovering Sysadmin
- Principal Security Consultant @ Semperis
- Open-source Toolmaker
- Microsoft MVP:
  PowerShell + Identity & Access

Agenda

- History & Goals
- Encoding vs. Encryption
- Symmetric & Asymmetric Encryption
- Hashing Functions
- Signing
- Certificates
- Public Key Infrastructure (PKI)
- Real-World Uses

Non-Agenda

- Cryptocurrency
- Key Exchange
- MAC/HMAC
- Algorithms
- Cryptanalysis
- Key Space
- Steganography
- Active Directory Certificate Services

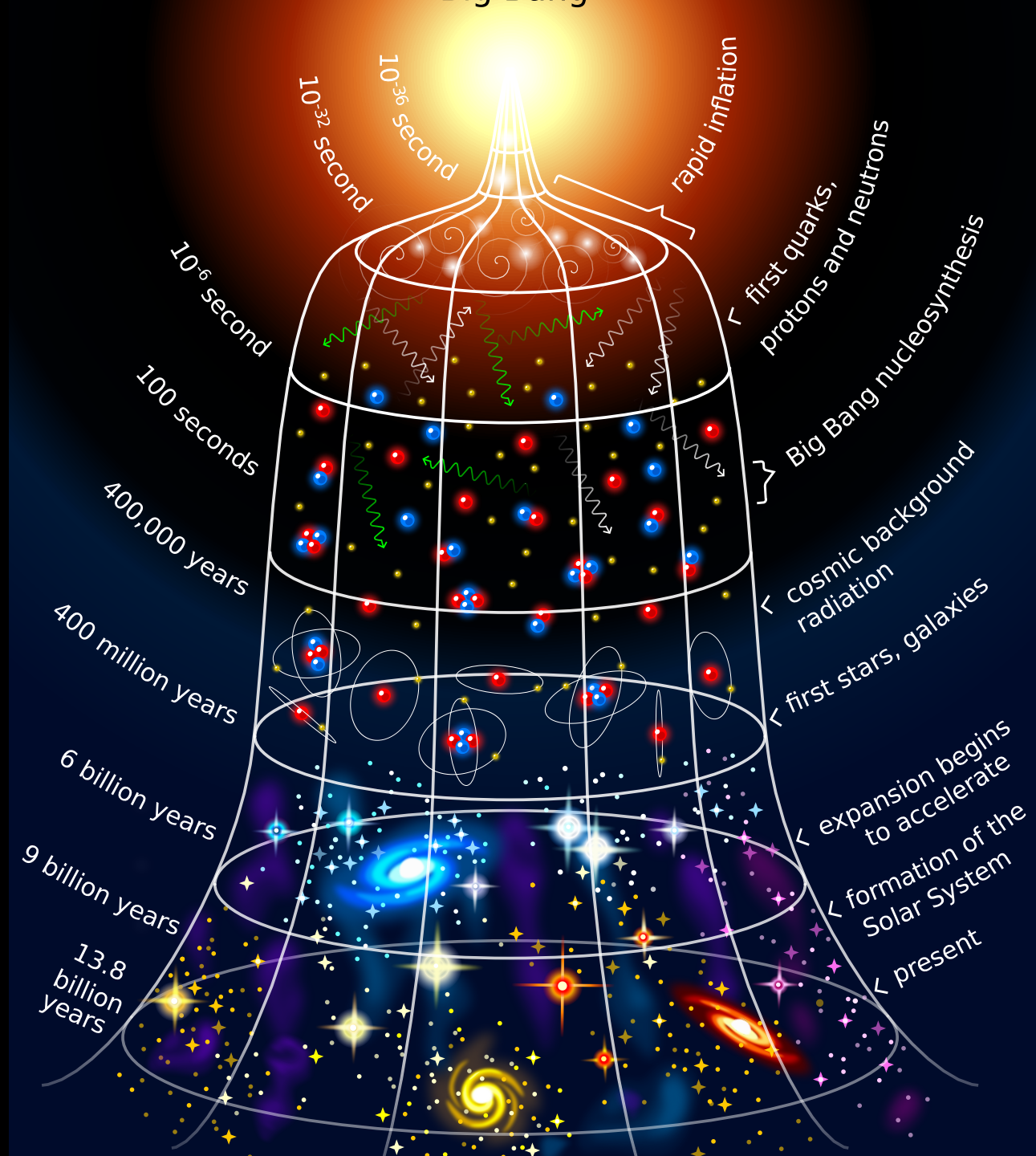
History

In the beginning was the word...

semperis

History

In the beginning was the **pass**word...



Antiquity

- 1900 BCE: Non-Standard Hieroglyphs
- 1500 BCE: Clay Tablets
- 600 BCE: Hebrew Scholars
- 400 BCE: Polybius Square
- 300 BCE: Kama Sutra
- 196 BCE: Rosetta Stone
- 75 BCE: Caesar Cipher

Medieval

- 750-800 CE: Arabic Scholars
- 800-1100 CE: English Scribes
- 1400 CE: Arabic Scholars
- 1500s CE: Tabula Recta/Vignère Cipher
- 1626 CE: Antoine Rossignol

Pre-WWII

- Focus on Cryptanalysis
- Many Medieval Systems Broken
- First Cryptographic Machines
- First One Time Pad

WWII

- Allies:
  - TypeX, SIGABA, Lacida
- Axis:
  - Enigma, Purple, SG-41

Modernity

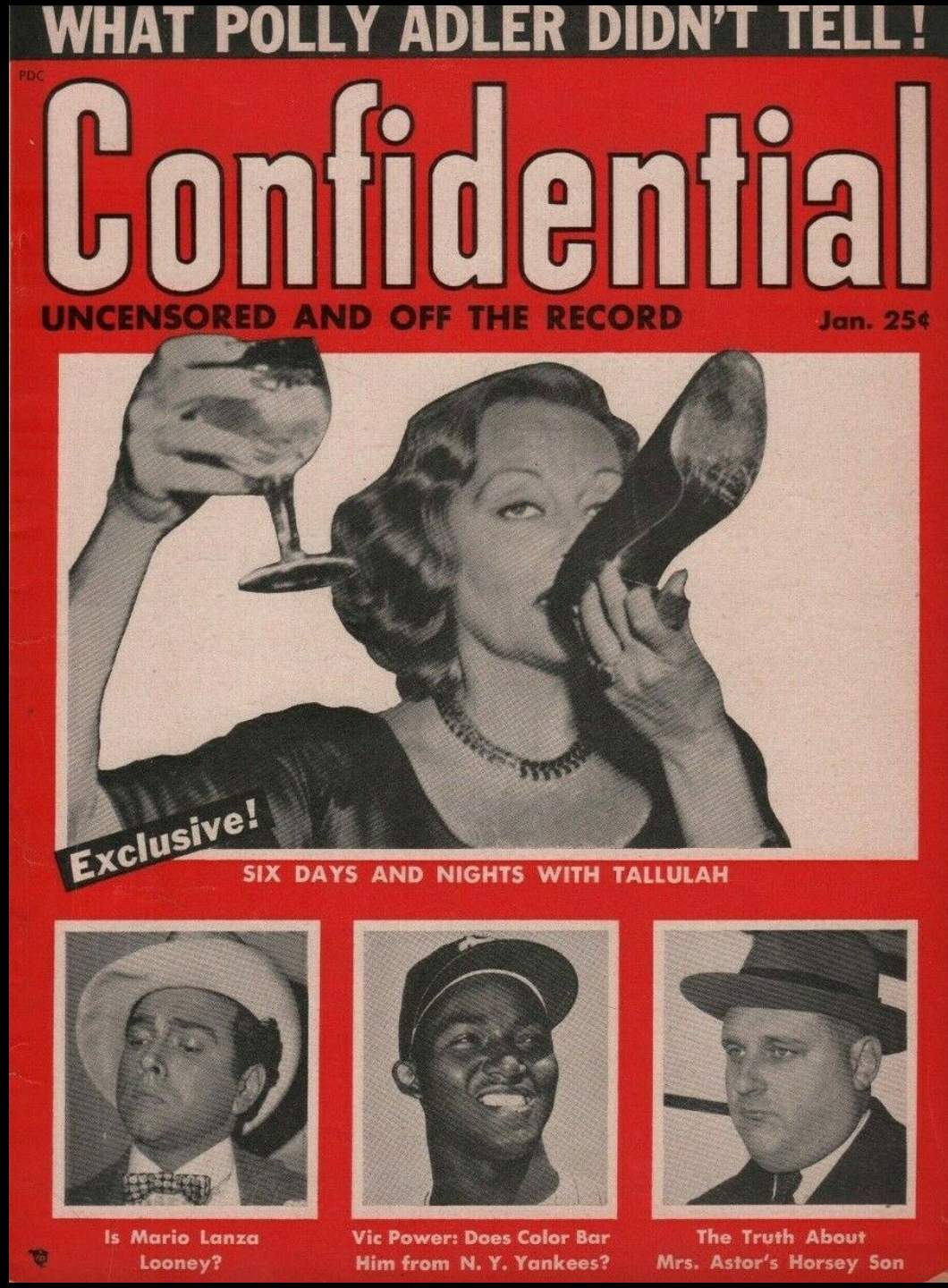
- 1950s: Claude Shannon
- 1975: Data Encryption Standard (DES)
- 1976: Diffie-Hellman (DH)
- 1977: Rivest, Shamir, Adelman (RSA)
- 1989: Message Digest Algorithms (MD*)
- 1991: Digital Signature Algorithm (DSA)
- 1995: Secure Hash Algorithms (SHA)
- 1985-2004: Elliptic Curve Cryptography (ECC)
- 2001: Advanced Encryption Standard (AES)
- The Future: Post-Quantum Cryptography (PQC)

Goals of Cryptography

semperis

Confidentiality

Ensures a message is only readable by its intended recipient



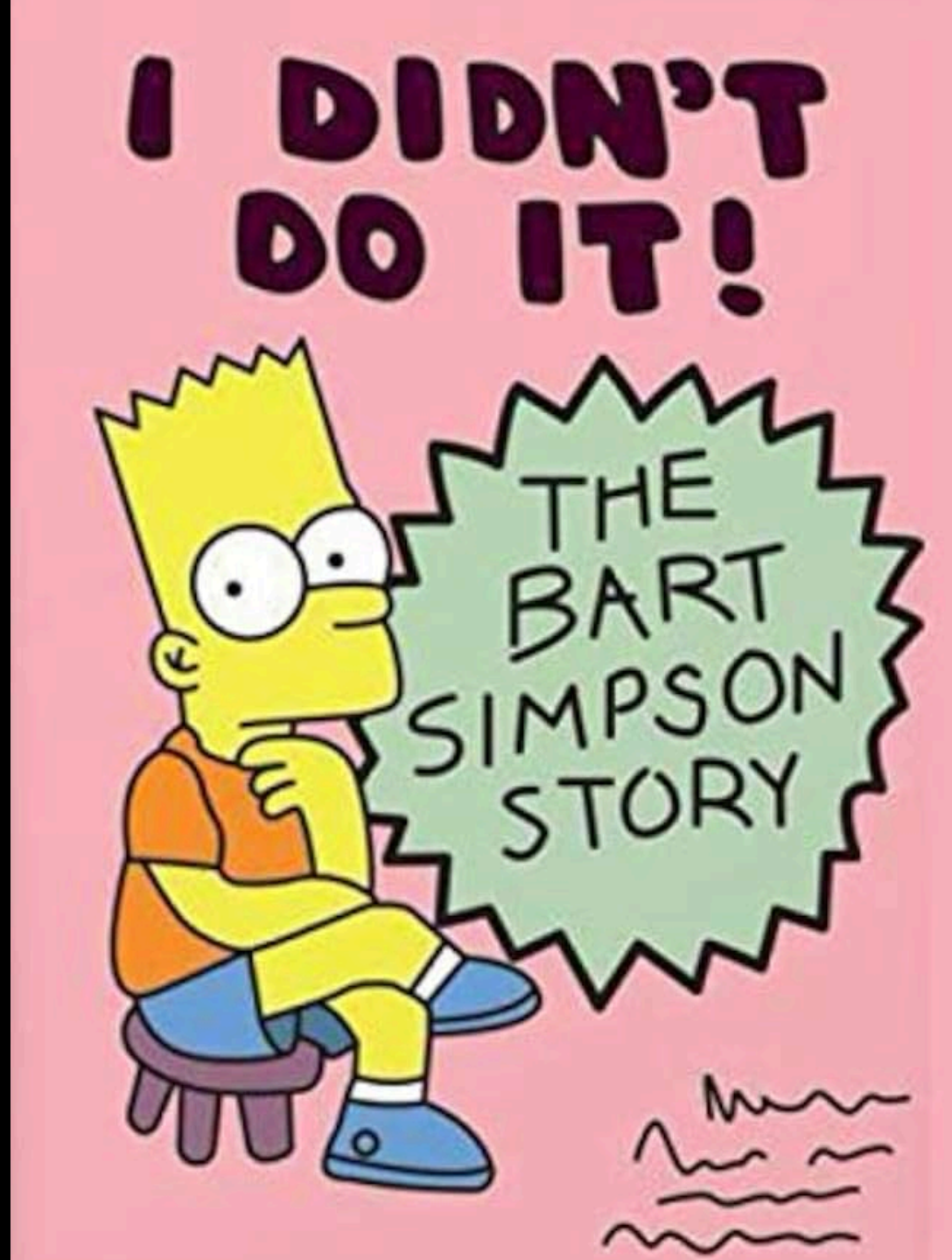
Integrity

Ensures a message has not been tampered with or changed

semperis

Non-Repudiation

Ensures an author cannot refute authorship of a message



Authentication

Provides proof an author of a message is who they claim to be

Encoding vs. Encryption

Secret or nah?

Encoding

- Not intended to remain secret
- that is, no **Confidentiality**

Common Schemes

- ASCII
- UTF-8/16/32
- Braille
- Morse Code
- Base64

semperis

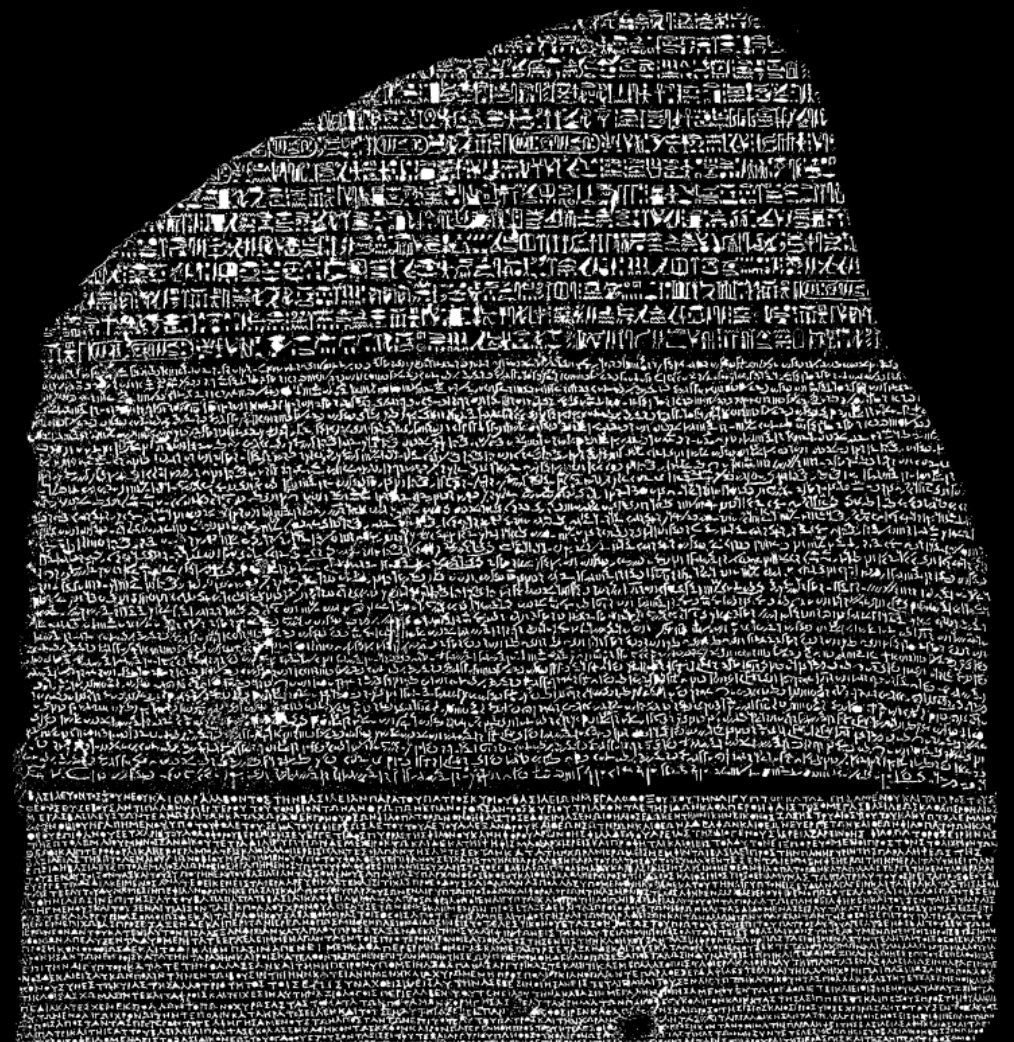
Example: Cetacean Cipher

- "Hello, CodeMash!" encoded:
- EEEEEEEEEeEEeEEEEEEEEEEEEeeEEeEe
  EEEEEEEEEeeEeeEEEEEEEEEEEeeEeeEE
  EEEEEEEEEeeEeeeeEEEEEEEEEeEeeEE
  EEEEEEEEEeEEEEeeEEEEEEEEEeeEeeee
  EEEEEEEEEeeEEeEEEEEEEEEEEeeEEeEe
  EEEEEEEEEeEEeeEeEEEEEEEEEeeEEEEe
  EEEEEEEEEeeeEEeeEEEEEEEEEeeEeEEE
  EEEEEEEEEeEEEEe

Try it yourself: www.a.tools/Tool.php?Id=389

Example: Rosetta Stone

- Ancient Egyptian was indecipherable
- Rosetta Stone showed Egyptian next to Greek
- Allowed for translating / "decoding" the Egyptian.
- Encoding can be mistaken for encryption - but once the method is known, it is trivial to break.



Encryption

- Intended to remain secret
- Yes! **Confidentiality**
- Requires 1 or more "keys"
- Keys are **something** used to encrypt/decrypt
- In modern cryptography, keys are **really** big numbers
- Kerckhoffs's principle

Common Schemes

- Symmetric - 1 key
  - DES
  - AES
- Asymmetric - 2+ keys
  - RSA
  - ECC

Encrypted data is

indistinguishable from random noise

Encrypted data ~~is~~ should be

indistinguishable from random noise


CodeMash 2026-01-15 · Jake Ireth · jake.ireth.com

Symmetric Encryption

Pretty much all cryptography before the 1970s

Symmetric Encryption

Pros

- Same key to encrypt/decrypt message
  - Easy to understand
- Simple algorithms
  - Very fast
- Provides **Confidentiality**

Cons

- Keys must be pre-shared via a secure channel before communication
  - Difficult!
- One key per communication group
  - Key management nightmare
- Does not:
  - protect **Integrity**
  - ensure **Non-Repudiation**
  - enable **Authentication**

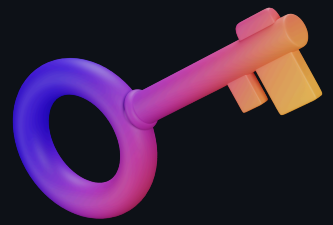
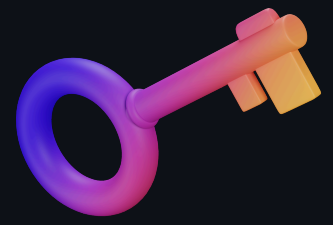
Example: Caesar Cipher (Symmetric)

- Also known as: Shift cipher or ROT*
- Symmetric Encryption ~= ROT13
- A(1) + Shift(13) = N(14)
- N(14) + Shift(13) = A(1)

Original	A	B	C	D	E	F	G	H	I	J	K	L	M
Value	1	2	3	4	5	6	7	8	9	10	11	12	13
"Encrypted"	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Original	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Value	14	15	16	17	18	19	20	21	22	23	24	15	26
"Encrypted"	A	B	C	D	E	F	G	H	I	J	K	L	M

Example: Caesar Cipher (Symmetric)

Original Message	Alice Encrypts (ROT13)	Encrypted Message	Bob Decrypts (ROT13)	Received Message
Hello, CodeMash!		Uryyb, PbqrZnfu!		Hello, CodeMash!

Try it yourself: https://rot13.com

Asymmetric Encryption

Encrypt with one key

Decrypt with a different key

Asymmetric Encryption

Pros

- One **key pair** (public + private) per user
  - Easy(ish) to manage
- Public keys can be shared over insecure channel
  - Much easier than secure channel
- Provides **Confidentiality**, **Integrity**, **Non-repudiation**

Cons

- Complicated algorithms
  - Not fast
- No **Authentication** of sender's identity
  - Impersonation is possible

Example: Caesar Cipher (Asymmetric)

- Private Key ~= ROT7
- Public Key ~= ROT19
- A(1) + Shift(7) = H(8)
- H(8) + Shift(19) = A(1)

Original	A	B	C	D	E	F	G	H	I	J	K	L	M
Value	1	2	3	4	5	6	7	8	9	10	11	12	13
"Encrypted"	H	I	J	K	L	M	N	O	P	Q	R	S	T

Original	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Value	14	15	16	17	18	19	20	21	22	23	24	15	26
"Encrypted"	U	V	W	X	Y	Z	A	B	C	D	E	F	G

Example: Caesar Cipher (Asymmetric)

Original Message	Alice Encrypts (ROT7)	Encrypted Message	Bob Decrypts (ROT19)	Received Message
Hello, CodeMash!		Olssv, JvklThzo!		Hello, CodeMash!

Try it yourself: https://rot13.com

Hash Functions

Neither potatoes nor pipes

Hash Functions

- Maps any data to a (hopefully) unique fixed-length value
- Can't be easily reversed to discover source data
- Changing even a single bit of source data changes the hash
- Ensures **Integrity**

Common Schemes

- MD4/5
- SHA-0/1/2/3
- DSA

semperis

Silly Example: Addition + LeftPad

Original	A	B	C	D	E	F	G	H	I	J	K	L	M
Value	1	2	3	4	5	6	7	8	9	10	11	12	13

Original	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Value	14	15	16	17	18	19	20	21	22	23	24	15	26

- HELLOCODEMAS*H*
  - 8+5+12+12+15+ 3+15+4+*5* =
  - Hash: **0079** or **00GI**
- HELLO
  - 8+5+12+12+15
  - Hash: **0052** or **00EB**
- HELLOCODEMAS*T*
  - 8+5+12+12+15+ 3+15+4+*20* =
  - Hash: **0094** or **00ID**

Note: This is a terrible hash function full of collisions. Do not use this for anything.

Real Example #1: Single Character Change

Original	SHA1
Hello, CodeMas*h*!	`4edf8a86e11c4adbf7839e9940666ce882f563cc`
Hello, CodeMas*t*!	`2d6ea1719370b90fab69e08bc5dea3bd7c77b4e6`

Try it yourself: https://gchq.github.io/CyberChef

Real Example #2: Variable-Length -> Fixed Length

Original	SHA1
The quick brown fox jumps over the lazy dog.	`408d94384216f890ff7a0c3528e8bed1e0b01621`
The quick brown fox jumps over the lazy dog. (x100)	`b09876f717b413763e960238754c6f22d2143ac2`
The quick brown fox jumps over the lazy dog. (x200)	`1f75bf99415e85c75fde244ec71de7f5e2d226ba`

Try it yourself: https://gchq.github.io/CyberChef

Signing

Hashing + Asymmetric Encryption = Better Than Ink

Signing Process

- Alice creates a message
- Alice creates a hash of her message
- Alice encrypts **the hash** with her **private** key
- Alice sends package to Bob
- Bob simultaneously:
  - decrypts hash w/ Alice's **public** key
  - hashes the original message
- If hashes match, Alice is the sender!

Common Uses

- Used for **Non-repudiation**
- Software distribution
- Financial transactions
- Contract management
- Network protocols

Example: Addition (Hashing) + Caesar Cipher (Asymmetric Encryption)

Original Message + Hash	Alice Encrypts Hash (ROT7)	Original Message + Encrypted Hash	Bob Decrypts Hash (ROT19)	Original Message + Decrypted Hash	Bob Hashes Message HImself
"Hello, CodeMash!" Hash: **00GI**		"Hello, CodeMash!" Encrypted Hash: **00TV**		"Hello, CodeMash!" Decrypted Hash: **00GI**	"Hello, CodeMash!" Hash: **00GI** MATCH!

Certificates

Public Key and Attributes, Signed… by someone!

Certificates

- Public keys don't include identifying information
- Certificates tell who owns a public key
- Certificates provide basic **Authentication**
- **If** you trust the issuer, you can trust the public key
- Self-signed certs still permit encryption!

Typical Contents

- Subject
- Issuer
- Public Key
- Not Before/Not After
- Key Usage/Extended Key Usage
- Signature Algorithm/Signature
- Serial Number

Generating a Self-Signed Certificate

Create a Key Pair	Create CSR	Hash the CSR	Encrypt Hash w/ Alice's Private Key	Package Certificate
 	Subject: Alice Public Key: 	0BHF	0UAY	Subject: Alice Issuer: **Alice** Public Key:  Signature Algorithm: Silly Signature: 0UAY

Certification Authority (CA) Certificate Generation

Create a Key Pair	Create CSR	Hash the CSR	Encrypt Hash w/ CA's Private Key	Package Certificate
 	Subject: Alice Public Key: 	0BHF	0KQO	Subject: Alice Issuer: **CA** Public Key:  Signature Algorithm: Silly Signature: 0KQO

Public Key Infrastructure (PKI)

Formalized trust - all the way down to the root

PKI Basics

- Not **just** technology:
  - Hardware
  - Software
  - Policies
  - Procedures
- "Chains" of trust lead back to a "root" of trust
- **If** you trust the PKI (big if), you can trust others that use the PKI

Common Components

- Authorities:
  - Root/Intermediate/Issuing
  - Validation
  - Registration
  - Policy
  - Timestamp
- Users:
  - End Entities
  - Relying Parties

How Do I Know If I Should Trust a PKI?

semperis

How Do I Know If I Should Trust a PKI?

How Do I Know If I Should Trust a PKI?

- Mostly handled for you
- OSes include lists of trusted roots
  - They're also usually out of date - Server 2025 GA included 7 Root CAs that were expired.
- Active Directory (AD) networks have PKI: AD Certificate Services
  - Drink!
- Reputable PKIs publish their configurations and procedures
- Honestly, something I need to dig into...

Real-World Uses

Hybrid Cryptosystems - **VERY** Simplified

Pretty Good Privacy (PGP)/GNU Privacy Guard (GPG)

Encryption Process

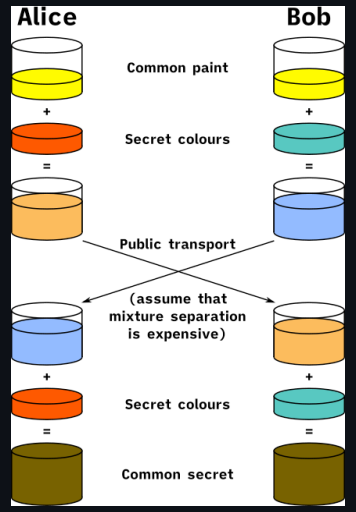
Alice Writes a Message & Creates a Session Key	Alice Encrypts the Message with the Session Key	Alice Encrypts The Session Key With Bob's Public Key	Alice Combines the Encrypted Message and Session Key And Sends to Bob
Message: "Hello, CodeMash!" Session Key: 4dyea1wo	Encrypted Message: "Fcjjm, YlrgAyqr!"	Encrypted Session Key: 7tjtf57d	Encrypted Message: "Fcjjm, YlrgAyqr!" Encrypted Key: 7tjtf57d

PGP/GPG

Decryption Process

Bob Receives the Encrypted Package	Bob Decrypts The Session Key With His Private Key	Bob Decrypts the Message with the Session Key
Encrypted Message: "Fcjjm, YlrgAyqr!" Encrypted Key: 7tjtf57d	Decrypted Key: 4dyea1wo	Decrypted Message: "Hello, CodeMash!"

Secure Sockets Layer (SSL)/Transport Layer Security (TLS)

Client Contacts Server via HTTPS	Server Sends Certificate for Client to Validate	Server and Client Agree on a Session Key (Diffie-Helman)	Data Stream is Encrypted with Session Key
dotdot.horse	Name: dotdot.horse Issuer: CA Signature: Signature		

Quick Review

Quick Review

- Humans have wanted to hide things **forever**
- Encoding is reversible **w/o keys**
- Encryption is reversible **with keys**
  - Symmetric Encryption:
    - Uses 1 key
    - FAST
  - Asymmetric Encryption:
    - Uses 2+ keys
    - SLOW
- Hashing is not reversible
- Signing combines hashing and encryption to provide non-repudiation
- Certificates bind an identity to a public key
- PKI solves a lot of problems… but at a cost
- Most modern cryptosystems combine symmetric and asymmetric encryption, hashing, signing, and certificates!

Resources

github.com/jakehildreth/CodeMash26

Links

Cetacean Cipher: a.tools/Tool.php?Id=389

CyberChef: gchq.github.io/CyberChef

Caesar Cipher: rot13.com

Thanks, CodeMash!

Find	Me
LinkedIn	linkedin.com/in/jakehildreth
GitHub	github.com/jakehildreth
Site	jakehildreth.com