Everyday Cryptography

Alexander Færøy

The Camp

July 26, 2016

Table of Contents

`/dev/random` and `/dev/urandom`.

Cryptographic Primitives

Bits of Security

- To help comparing systems and pick "compatible" building blocks, we measure cryptosystems in bits of security.
- The number of bits is used to describe the amount of work an adversary will have to complete before they will be able to compromise or weaken our system.

Cryptographic Primitives

Xor

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

Cryptographic Primitives

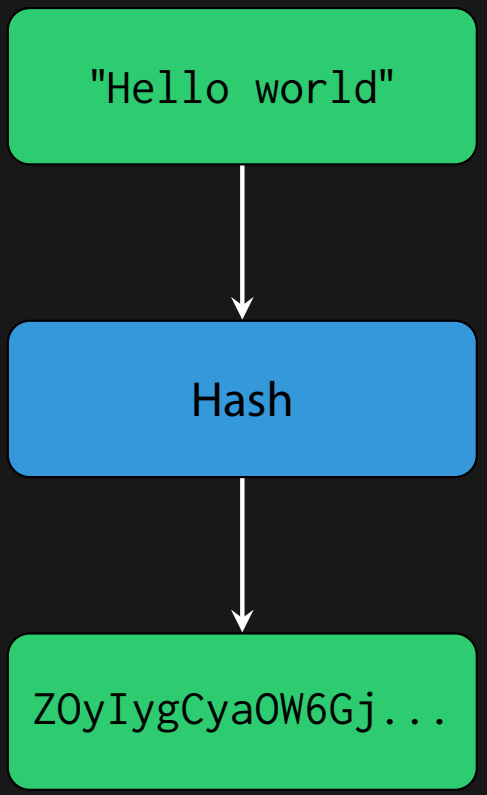
Xor

"foobar" $\oplus$ "abcdef" = $7, 13, 12, 6, 4, 20$

$7, 13, 12, 6, 4, 20$ $\oplus$ "abcdef" = "foobar"

Hash Functions

- A cryptography hash function is a function that takes an arbitrary length input and returns a fixed length output.
- Computing $y = f(x)$ should be "fast", but computing the inverse function $x = f'(y)$ must be NP-hard.
- Even minor, single bit, modifications to the input must yield a different output value.
- As of 2016, we mostly use SHA-2 and newer functions in our protocols.
- Do NOT use MD5, RIPEMD-160, and SHA-1 anywhere.
- For new code: Use SHA-3.



Passphrase Hashing

- Commonly used for passphrase storing and for deriving keys for disk encryption.
- Currently very popular algorithm is: PBKDF2 with a salt.
- Modern choices: Argon2 and scrypt.
- One must be careful when making the choice of function since DoS attacks might become possible.

https://password-hashing.net/

Secret-Key Algorithms

- A crypto system where both parties must agree on a key before they can exchange encrypted data.
- The oldest cryptography systems we have.

Camp Cipher

```
$ python
>>> import camp
>>> camp.encrypt("foobar", "Hello world!")
'LgoDDg5SEQAdDgVT'
>>> camp.decrypt("foobar", "LgoDDg5SEQAdDgVT")
'Hello world!'
```

Camp Cipher

```
import base64

def stream(key, message):
    key_length = len(key)
    i = 0
    r = ""

    for char in message:
        r += chr(ord(char) ^ ord(key[i % key_length]))
        i += 1

    return r

def encrypt(key, message):
    return base64.b64encode(stream(key, message))

def decrypt(key, message):
    return stream(key, base64.b64decode(message))
```

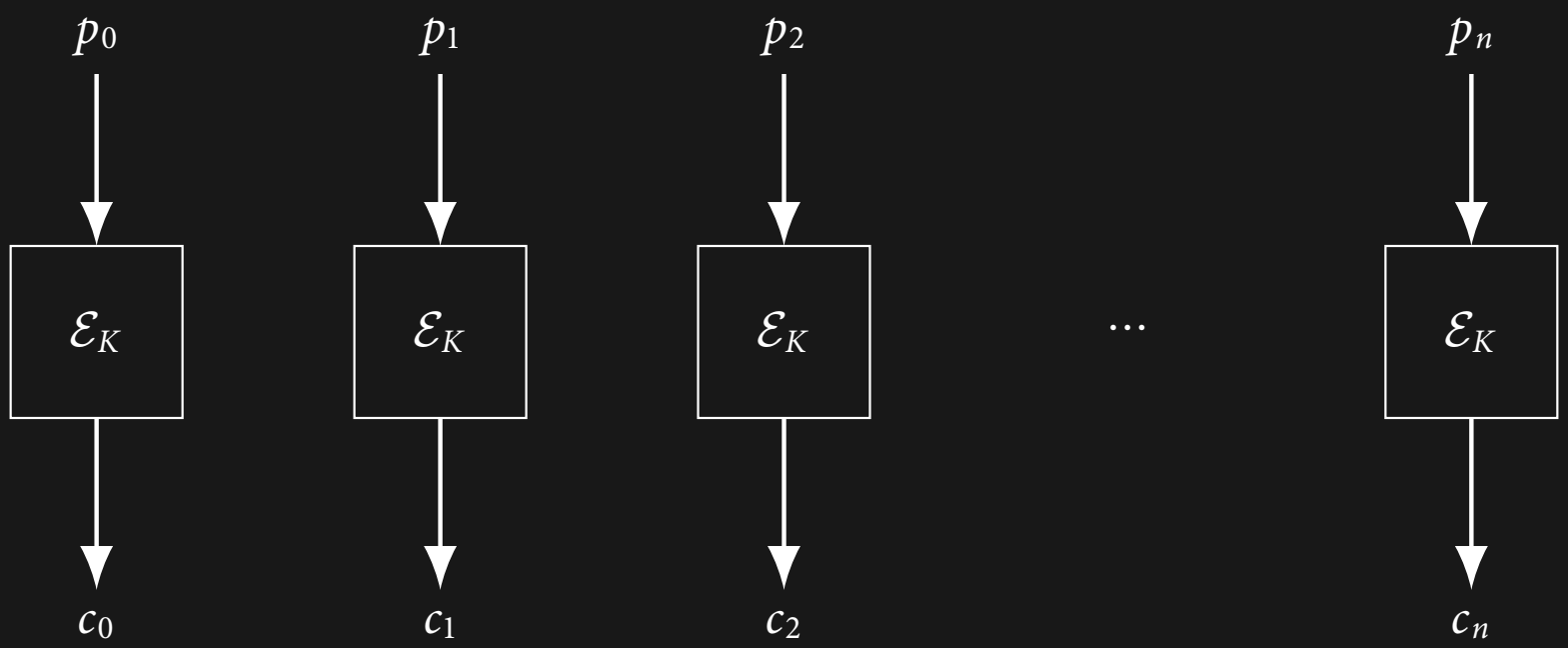
Advanced Encryption Standard

- Originally named Rijndael after its inventors Joan Daemen and Vincent Rijmen.
- Published in 1998.
- Won the AES competition in 2001 and thus superseded the Data Encryption Standard.
- Worth mentioning: Serpent.
- CPU support in modern Intel CPU's. Allows us to do around 1 GB/sec of AES.

Joan Daemen and Vincent Rijmen.

Advanced Encryption Standard

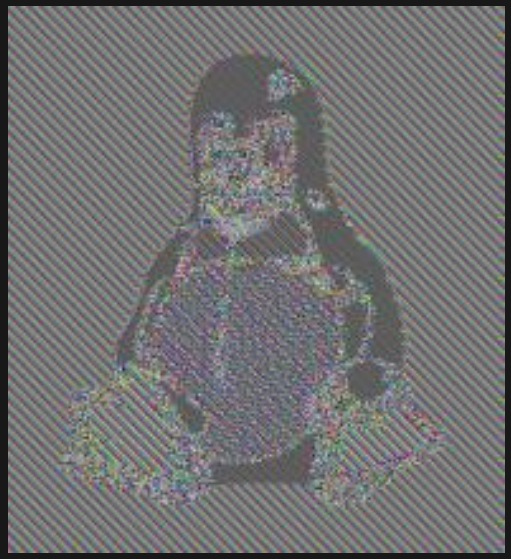
ECB Mode



Advanced Encryption Standard

ECB Mode

Original image.

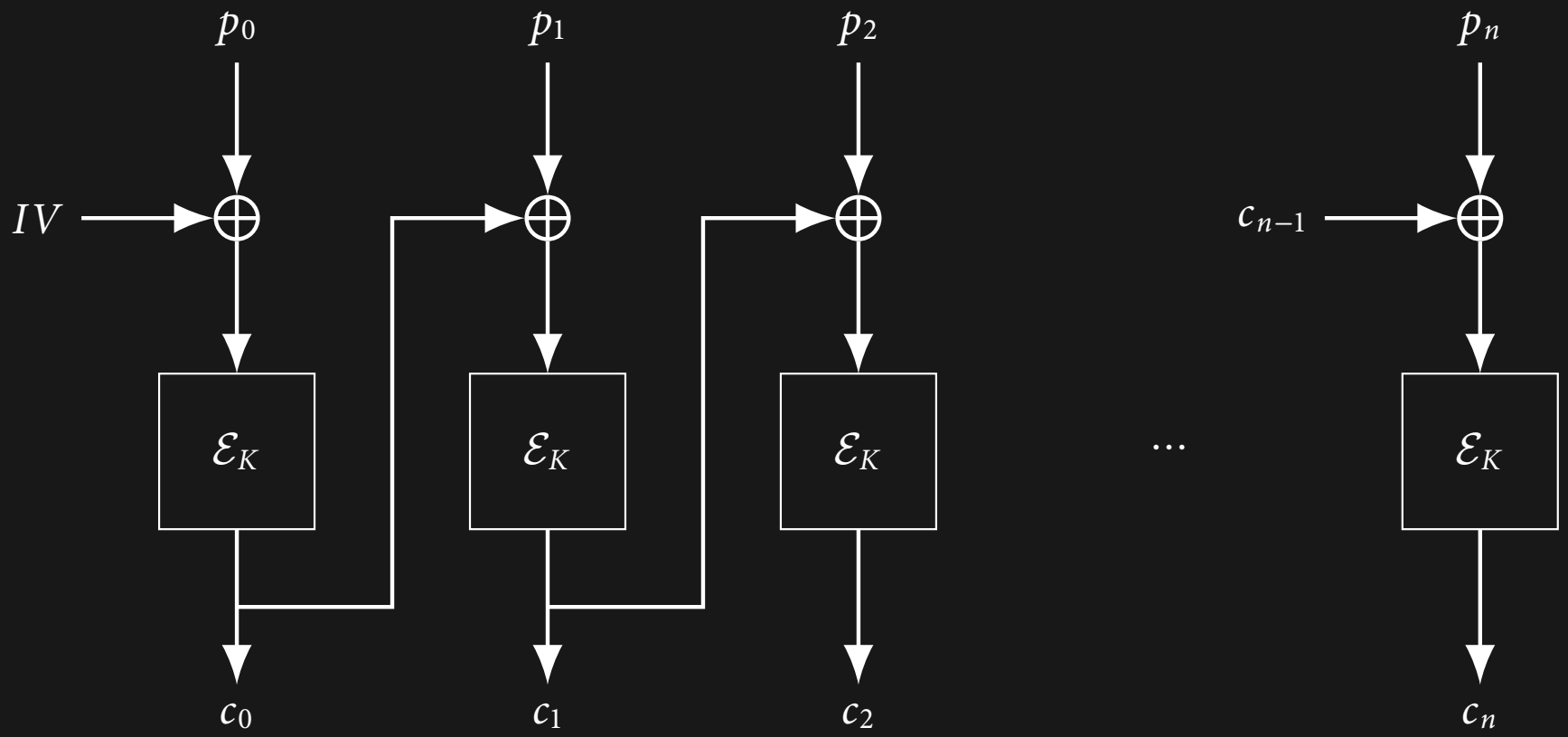


Encrypted with ECB mode.

Encrypted with a secure mode.

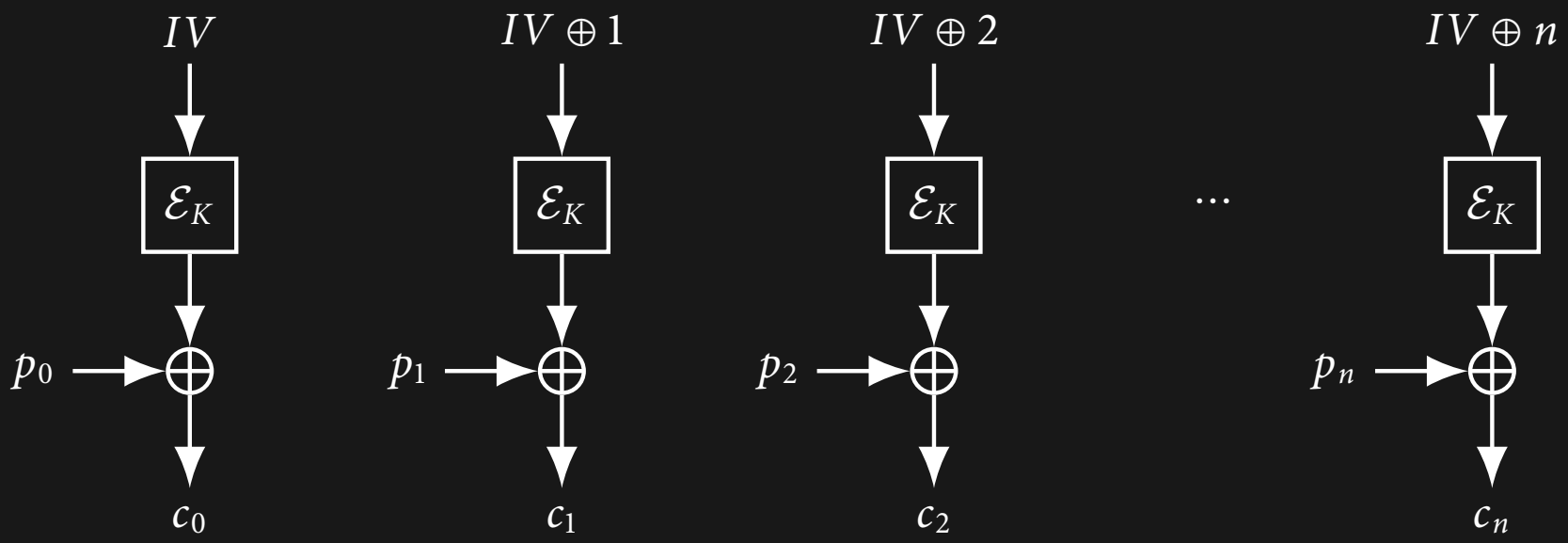
Advanced Encryption Standard

CBC Mode



Advanced Encryption Standard

CTR Mode

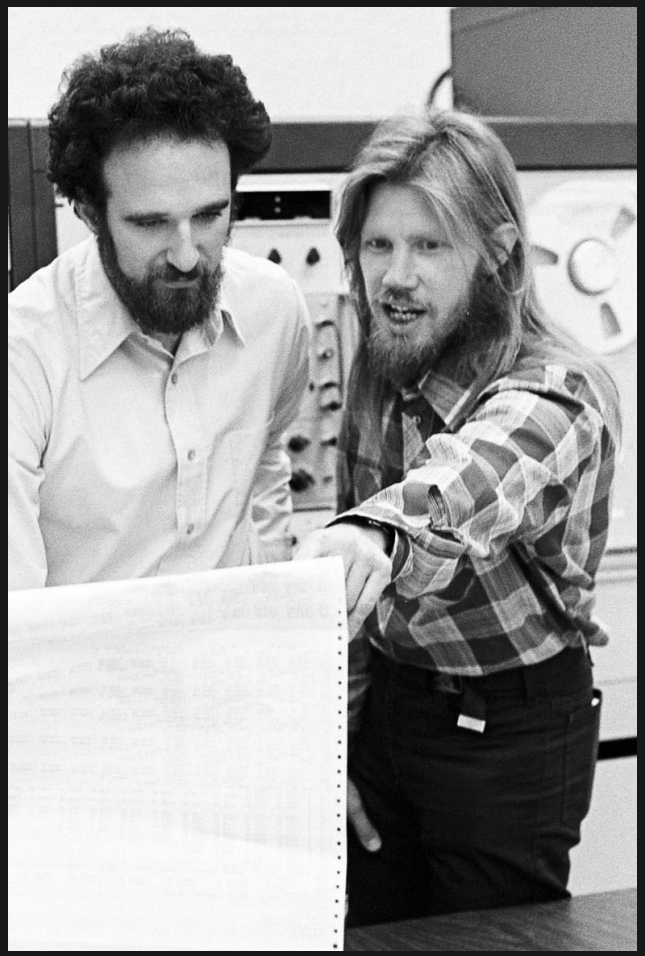


Public-Key Algorithms

- Each party have two keys: a secret key and a public key. The public key can be published, the secret key must remain secret.
- Was a major milestone in the 70ies where it became possible.
- Often used together with secret key algorithms.

Diffie-Hellman Key Exchange

- Diffie-Hellman key exchange is an algorithm for establishing a secret key over an insecure, public, communication channel.
- The cryptosystem was published by Whitfield Diffie and Martin E. Hellman in 1976, but was discovered by James H. Ellis, Clifford Cocks, and Malcolm J. Williamson from GCHQ in 1975, but the discovery was kept secret until 1997.
- Uses the discrete log problem: $g^x \pmod{p}$ is hard to reverse.
- Whitfield Diffie and Martin E. Hellman was awarded the A.M. Turing Award in 2015 for their work on cryptography.

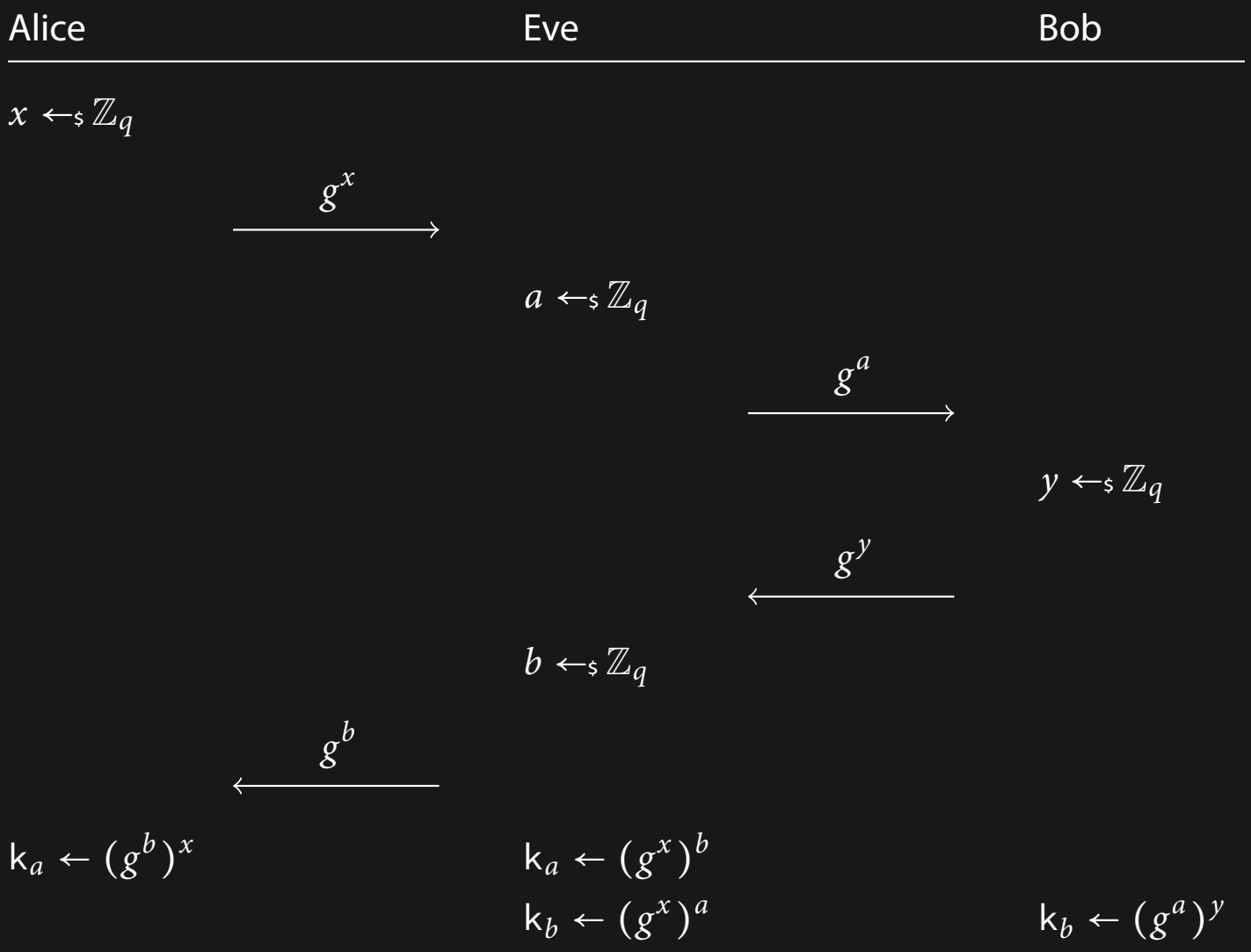


Martin E. Hellman and Whitfield Diffie.

Diffie-Hellman Key Exchange

Alice		Bob
$x \leftarrow_{\$} \mathbb{Z}_q$		
	$g^x \longrightarrow$	
		$y \leftarrow_{\$} \mathbb{Z}_q$
	$\longleftarrow g^y$	
$\mathsf{k} \leftarrow (g^y)^x$		$\mathsf{k} \leftarrow (g^x)^y$

Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange

```
import random

class DH(object):
    def __init__(self, g, p):
        self._g = g
        self._p = p

    def keypair(self):
        secret = random.randint(self._g + 1,
                                self._p - 1)
        public = pow(self._g, secret, self._p)
        return (secret, public)

    def shared(self, secret, public):
        return pow(public, secret, self._p)
```

Diffie-Hellman Key Exchange

```
$ openssl dhparam -out params.pem 2048
Generating DH parameters, 2048 bit long safe prime,
generator 2
This is going to take a long time
.....................................++++++*
```

Diffie-Hellman Key Exchange

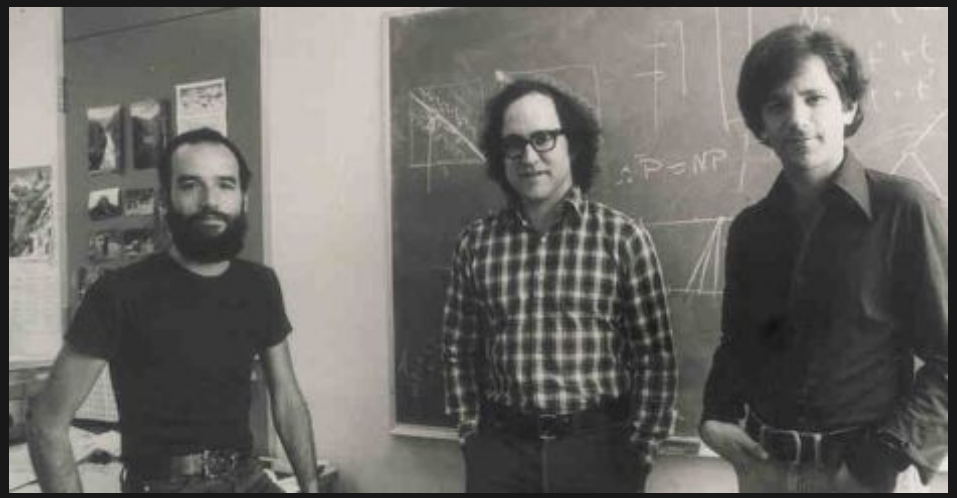
```
$ openssl dhparam -out params.pem 2048
Generating DH parameters, 2048 bit long safe prime,
generator 2
This is going to take a long time
.....................................++++++*
$ openssl asn1parse -in params.pem
    0:d=0  hl=4 l= 264 cons: SEQUENCE
    4:d=1  hl=4 l= 257 prim: INTEGER   :E490 ...
  265:d=1  hl=2 l=   1 prim: INTEGER   :02
```

Diffie-Hellman Key Exchange

```
$ openssl dhparam -out params.pem 2048
Generating DH parameters, 2048 bit long safe prime,
generator 2
This is going to take a long time
.....................................+++++*
$ openssl asn1parse -in params.pem
    0:d=0  hl=4 l= 264 cons: SEQUENCE
    4:d=1  hl=4 l= 257 prim: INTEGER   :E490 ...
  265:d=1  hl=2 l=   1 prim: INTEGER   :02
$ python3 dh.py 02 E490 ...
Alice's Public Key: 3276 ...
Bob's Public Key:   2522 ...
Shared Secret:      7481 ...
```

RSA

- Published in 1977.
- Supports both encryption and signatures.
- Usually combined with secret key cryptosystems.
- Rivest, Shamir, and Adleman was awarded the A.M. Turing Award in 2002 for their work on public-key cryptography.
- Large keys: 2048 to 4096 bits today.
- Build using prime factorization.



Adi Shamir, Ronald L. Rivest, and Leonard M. Adleman.

Curve25519 Key Exchange

- Published in 2006 by Daniel J. Bernstein.
- ECDH function used to establish a secure channel over an insecure connection.
- 32 bytes public keys, 32 bytes secret keys, 32 bytes shared secrets.
- Key generation is very simple:

```
secret = os.urandom(32)
secret[0] &= 248
secret[31] &= 127
secret[31] = 64
```

Daniel J. Bernstein.

https://cr.yp.to/ecdh.html

Ed25519

- Publised in 2011 by Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang.
- Signatures are 64 bytes, public keys are 32 bytes, secret keys are 64 bytes.
- Provides 2^{128} bits of security. Equivalent to a 3000 bits RSA key.
- Can generate around 100.000 signature signatures per second.
- Verification performance is bounded by hashing.
- Key generation performance is partially bounded by the time it takes to read from `/dev/urandom`.
- Designed with focus on not relying on branch conditions when working with the secret key.

Daniel J. Bernstein.

Tanja Lange.

http://ed25519.cr.yp.to/

Ed25519

```
$ gpg2 --fingerprint 0xE15081D5D3C3DB53
pub   ed25519/0xE15081D5D3C3DB53 2015-02-14 [SC] [expires: 2020-02-28]
      Key fingerprint = E265 2A9B 5D17 14B5 5CE3  ADE1 E150 81D5 D3C3 DB53
uid                 [ultimate] Alexander Faeroey (Code Signing Key) <ahf@0x90.dk>
```

Pretty Good Privacy

- Published in 1991 by Phil Zimmermann.
- Supports RSA, DSA, ElGamal, Ed25519, and AES.
- Verification using web-of-trust and by having people attending keyparties.
- Used for email encryption and authentication and package authentication.
- Was in the 90ies affected by the US export laws on cryptography.

Philip R. Zimmermann.

https://gnupg.org/

Transport Layer Security

1. Client connects over TCP to a server, sends random number, and a list of supported cipher suites.
2. Server sends random number, public key and certificate.
3. If supported, the server sends the DH parameters.
4. Client sends random session key, encrypted to the server's public key.
5. Client sends DH parameters.
6. Moves to secret key communication.

OpenSSL and LibreSSL.

Off The Record

- End-to-end Encryption for instant messaging (Jabber, Facebook, Google Chat).
- Provides encryption, authentication, deniability, and forward secrecy.
- Authentication is done with fingerprint verification (like with PGP) or Socialist Millionaires' Protocol.
- Uses Discrete Log Diffie-Hellman, AES-128, DSA, SHA-1 and SHA-256.
- Work is being done on moving to Ed25519, Curve25519 and a stronger hash function.

Ian Avrum Goldberg.

https://otr.cypherpunks.ca/

Signal

- Uses Curve25519, 3-DH, HMAC-SHA256 and AES.
- Provides forward secrecy and backward secrecy.
- The protocol was originally named Axolotl.
- No focus on anonymity.

Moxie Marlinspike.

Post-quantum Cryptography

- Cryptographers have recently started focusing on cryptography that will work in a world where an adversary have access to a quantum computer.
- RSA is dead, DSA is dead, elliptic and hyperelliptic curves are dead, and discrete logarithm Diffie-Hellman is dead.
- Secret key cryptography still works.
- Weakest link in a chain: If your handshake was made using DH and you use the key for a secret key system it will be dead too.
- Current PGP, Tor, TLS, and Signal will be affected.

https://pqcrypto.org/

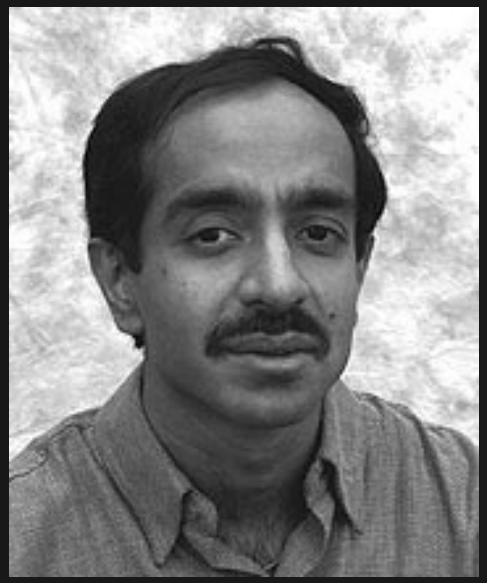
Shor's Algorithm

- Formulated in 1994 by Peter Shor.
- Quantum algorithm for integer factorization: "given an integer, N find its prime factors".
- In 2012, researches was able to factorize 21 into 3 and 7.
- Will break current RSA usage, if we have access to a 2048 to 4096 qubit quantum computers.

Peter Williston Shor.

Grover's Algorithm

- Formulated in 1996 by Lov Grover.
- Probalistic quantum algorithm for finding the unique input to a given blackbox function.
- Classical computers can do this operation in $\mathcal{O}(N)$ and due to Grover's algorithm, quantum computers will be able to do it in $\mathcal{O}\left(N^{1/2}\right)$.
- Will weaken secret key cryptography like AES, by reducing the security.



Lov Kumar Grover.

SPHINCS-256

- Published by Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn in 2014.
- Very large signatures: 41.000 bytes, public and secret keys are at 1024 bytes each.
- Aims at providing 2^{128} bits of post-quantum security.
- Can be used as a drop in replacement for already established systems. For example RSA or ed25519 signatures.
- Is able to sign hundreds of messages per second on a 4 core Intel CPU as of 2014.

https://sphincs.cr.yp.to/

New Hope

- Published by Erdem Alkim, Leo Ducas, Thomas Poppelmann, and Peter Schwabe in 2015.
- Diffie-Hellman like key exchange function, but requires one extra round-trip than ordinary DH and curve25519.
- Aims at providing 2^{128} bits of post-quantum security, with a "comfortable margin".
- 1824 bytes public keys, 1792 bytes secret keys, 32 bytes for shared secret.
- Proposed for new Tor handshake: 270 - RebelAlliance: A Post-Quantum Secure Hybrid Handshake Based on NewHope, by Isis Lovecruft and Peter Schwabe.
- Google's BoringSSL added the cipher suite CECPQ1 in May 2016, which is based on New Hope and curve25519.

https://github.com/tpoeppelmann/newhope

Questions?

This work is licensed under a

Creative Commons
Attribution-ShareAlike 4.0 International License