Digital safety relies on cryptography

STRONG CRYPTOGRAPHY IS CORE OF SECURITY

Confidentiality

Integrity

Authenticity
What if foundation becomes in-secure?
Impact of weak crypto is material

IMPACT OF WEAK CRYPTOGRAPHY

- Risk: Financial Loss
- Risk: Public Safety
- Risk: Data Loss
- Impact: Compliance

CRYPTO VULNERABILITY | SHALL IMMEDIATELY BE FIXED
Cryptographic failures
Malware Flame

MD5 Certificates

- Discovered 2012, attacks Windows
- Components of Flame were signed with a fraudulent certificate, made it appear to have originated from Microsoft
- Specific Microsoft certificate still used MD5 as hash function
- Malware authors produced a counterfeit copy of the certificate
- Rogue certificates demonstrated in 2008 by Sotirov et.al.
- MD5 is a cryptographic hash function published in 1992 and broken in 2004
DUAL_EC_DRBG

RNG with backdoor

- Algorithm that was presented as a cryptographically secure pseudorandom number generator
- Standardized by ANSI, NIST and ISO
- RSA made Dual_EC_DRBG the default CSPRNG in BSAFE
- Juniper were using it in their VPN products
- Contains a backdoor
• SHA-1 is a cryptographic hash function
• SHA-1 theoretically broken in 2005
• Since 2005 constant progress towards a practical attack
• Officially deprecated by NIST in 2011
• SHA-1 is used everywhere
  ▪ Digital Certificate signatures
  ▪ Email PGP/GPG signatures
  ▪ Software signatures
  ▪ Backup systems
  ▪ etc...

https://shattered.io
WEP

- WEP introduced in 1997 in 802.11 wireless standard
- Uses weak stream cipher (RC4) and short IVs
- First attacks in 2001 by various researchers
- Since then constant improvements have been published

- WEP was blamed for the theft of 45 million credit card numbers in 2007

Lucky 13
Padding oracle attack on TLS

- Attack on TLS up to version 1.2 published in 2013
- Attacks apply to CBC-mode in all TLS and DTLS implementations
- Issue with the order MAC-Encode-Encrypt, padding not included in MAC
- Man-in-the-middle attacker who sees only ciphertext and can inject ciphertexts of his own composition into the network
- Allows plaintext recovery in certain circumstances

http://www.isg.rhul.ac.uk/tls/Lucky13.html
Keeloq
Proprietary block cipher

- Is/was used in many remote keyless entry systems for cars
- Details of the algorithm were leaked in 2006
- Researchers broke the system in 2007
  - Recover 64-bit Keeloq key using only $2^{16}$ known plaintexts and $2^{44.5}$ encryptions
- In 2008 independent researchers show improved results
  - Their attack works on all known car and building access control systems that rely on the Keeloq cipher
  - Recovers the secret master keys embedded in both the receiver and the remote control using side-channel attacks

https://www.cosic.esat.kuleuven.be/keeloq/
http://www.emsec.rub.de/keeloq
ROCA
Vulnerable RSA generation

- Faulty generation of RSA keys used in cryptographic smartcards, security tokens and other secure hardware chips manufactured by Infineon Technologies AG
- Code complies with two security certification standards, NIST FIPS 140-2 and CC EAL5+,
- Allows for a practical factorization attack: remote attacker can compute an RSA private key from the public key

Estonia announced that 750,000 government identify cards, issued since October 2014, might be effected

- Vulnerable keys in TLS certificates discovered
- PGP keys used for email encryption effected
- Keys on GitHub submissions
- Etc…

How does crypto evolve?
Constant progress in cryptography
1970-1999

1974: Development of Feistel network

1976: DES published as FIPS standard

1977: Invention of RSA

1978: Invention of the McEliece cryptosystem

1979: A5/1 cipher for GSM encryption

1980: A5/2 developed as weakened variant of A5/1

1985: Miller and Koblitz discover elliptic curve cryptography

1987: A5/1 cipher for GSM encryption

1989: MD2 hash function published

1990: MD4 hash function published

1991: RSA-100 factored

1991: Phil Zimmermann releases PGP

1992: Rivest publishes the MD5 hash function

1992: SHA-0 standardized (FIPS 180)

1992: SHA-1 standardized (FIPS 180-1)

1993: DSA standardized (FIPS 186)

1994: SSL protocol released

1994: DES theoretically broken

1994: SHA-1 standardized (FIPS 180-1)

1995: First full-round collision attack on MD4

1996: Collision on MD5 compression function

1997: NIST launches AES competition

1998: Triple-DES published

1998: DES practically broken

1999: A5/2 broken

1999: RSA-155 factored

1999: A5/3 released for GSM and GPRS

1999: A5/3 released for GSM and GPRS

2000
Constant progress in cryptography

2000-2017

2000: RSA released into public domain
2000: ECDSA standardized (FIPS 186-2)
2000: Real-time cryptanalysis of A5/1
2001: AES standardized (FIPS 197)
2001: SHA-2 standardized (FIPS 180-2)
2003: CCM mode published
2005: Boomerang attack on A5/3
2005: RSA-640 factored
2006: DES broken in 9 days
2007: SHA-3 competition announced
2007: NIST includes GCM and CCM in SP 800-38D
2009: RSA-768 factored
2010: Practical related-key attack on A5/3
2012: SHA-512/224 and SHA-512/256 standardized (FIPS 180-4)
2013: Dual_EC_DRBG discovered to have backdoor
2015: SHA-3 standardized (FIPS 202)
2016: NIST announces PQC competition
2017: Practical collision attack on SHA-1
2017: DES broken in 25 seconds
Constant progress in cryptography

2017-20??

- IoT Standards and Regulations
- Functional Encryption
- Homomorphic Encryption
- Post-Quantum Cryptography Standards
- Lightweight Cryptography Standards
- Multi-Party Computation
- SHA-2 broken?
- Forged SHA-1 Certificates?
- RSA and DSA broken?
How diverse is crypto?
Many options

<table>
<thead>
<tr>
<th>Traditional cryptography</th>
<th>Lightweight cryptography</th>
<th>National cryptography</th>
<th>Post-Quantum Cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>CLEFIA</td>
<td>USA Suite A</td>
<td>Code-Based</td>
</tr>
<tr>
<td>ECC</td>
<td>PRESENT</td>
<td>GER ECGDSA</td>
<td>Hash-Based</td>
</tr>
<tr>
<td>AES</td>
<td>PHOTON</td>
<td>RU GOST</td>
<td>Isogeny-Based</td>
</tr>
<tr>
<td>Serpent</td>
<td>SPONGENT</td>
<td>KOR KCDSA</td>
<td>Lattice-Based</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Multivariate-Based</td>
</tr>
</tbody>
</table>

...
Why do we care?
Need for Agility

**Compliance**
- National crypto requires that hardware vendors comply to local algorithms and standards
- Supply Chains are managed globally, management of critical systems reside in one country and deployed globally
- Crypto agility required to comply with country standards

**Longevity of IoT**
- IoT devices in operation for 10+ years
- Often without possibility to update critical components
- Crypto agility required to adapt to new threats and new standards during the whole life cycle

**Cryptographic Threats**
- Vulnerable implementations
- Outdated algorithms
- New attacks on existing algorithms
- Crypto agility required to quickly react on these threats

**Future Cryptography**
- Post Quantum Cryptography
- New algorithm and standards
- New use cases
- Crypto agility required to be prepared when standards are announced
What is the solution?
We need crypto agility
Cryptographic agility is the ability of a system to easily adopt alternatives to the cryptographic primitives it was originally designed to use.
Cryptographic Agility

Decouple applications from cryptography
Agile Crypto Library
Abstract, Dynamic and Manageable

- **Abstract API**
  Hide crypto complexity to developers

- **Dynamically Loadable**
  Change crypto during runtime

- **Select Implementation Type**
  Depending on use case

- **Deploy New Algorithm**
  Without modifying application code

- **Let experts decide which crypto to use**
  Make it manageable by others

- **Make it run on everything**
  Support as many platforms as possible

SOFTWARE
HARDWARE
How could it look like?

Portability

GenerateKey() {
    provider = LoadProvider("path to implementation");
    error = GenerateSymKey(provider, key);
    if(error)...
    return key;
}

Encrypt(data, key) {
    provider = LoadProvider("path to implementation");
    error = Encrypt(provider, data, key, ciphertext);
    if(error)...
}
Existing Libraries?
Many open source libraries

How agile are they?

- OpenSSL
- BouncyCastle/Java Cryptographic Architecture
- libsodium
- Crypto++
- wolfSSL
- libgcrypt
- Network Security Services
- …
OpenSSL
EVP interface

- EVP functions provide a high level interface to OpenSSL cryptographic functions
- Support for an extensive range of algorithms
  - Encryption/Decryption
  - Sign/Verify
  - Key derivation
  - Secure Hash functions
  - Message Authentication Codes
  - Support for external crypto engines

https://wiki.openssl.org/index.php/EVP
EVP symmetric encryption and decryption

```c
EVP_CIPHER_CTX *ctx;
int len;
int ciphertext_len;

if(!(ctx = EVP_CIPHER_CTX_new())) handleErrors();

if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv)) handleErrors();

if(1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len)) handleErrors();
ciphertext_len = len;

if(1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len)) handleErrors();
ciphertext_len += len;

EVP_CIPHER_CTX_free(ctx);
```

https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption
OpenSSL

Engine interface

- Exposes an *Engine API*, which makes it possible to plug in alternative implementations of some or all of the cryptographic operations implemented by OpenSSL
- Usually used for cryptographic acceleration using a hardware device
- Engines can dynamically be loaded at runtime

- Example: PKCS11 engine
- Can only replace already existing algorithms
- Inherits limitations from EVP interface
OpenSSL

How agile is it?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation independence</td>
<td>no</td>
</tr>
<tr>
<td>Implementation simplicity*</td>
<td>mediocre</td>
</tr>
<tr>
<td>Implementation abstraction</td>
<td>mediocre</td>
</tr>
<tr>
<td>Dynamic exchangeability and extensibility</td>
<td>no</td>
</tr>
<tr>
<td>Manageability</td>
<td>no</td>
</tr>
<tr>
<td>Portability</td>
<td>good</td>
</tr>
<tr>
<td>Performance</td>
<td>good</td>
</tr>
</tbody>
</table>

*depends on personal preference and experience
Java Cryptography Architecture

Abstraction in Java

- Framework for cryptography in Java
- Provider-based architecture
- Abstract APIs for cryptographic operations
- Abstract data structures for cryptographic objects

- Separates the interfaces and generic classes from their implementations
Java Cryptography Extension

Abstraction in Java

- Part of JCA
- Provides encryption, key generation, key agreement and MAC
- In the past JCA and JCE used to be treated differently by US export policies

- Restrictions
  - Oracle JDK requires each provider to be signed by Oracle
  - OpenJDK does not
  - Keys > 128 bits require *Unlimited Strength Jurisdiction Policy Files* (default since 8u161)
Providers

Abstraction in Java

- Providers implement the API defined in JCA and JCE, and they are responsible for providing the actual cryptographic algorithm
- Popular providers: BouncyCastle, IAIK-JCE
- Can be configured through the Java security file

CRYPTOGRAPHIC API'S

Application

Providers

BouncyCastle

IAIK-JCE

...
Java Cryptography Architecture

Abstract API

- Encryption example:

```java
Security.addProvider(provider_name);
Cipher c = Cipher.getInstance("AES");
c.init(ENCRYPT_MODE, key);
byte[] cipherText = c.doFinal(plainText);
```
BouncyCastle

Provider

- Provider for JCA/JCE
- Includes also a low-level API (no restrictions)

```java
PBEKeySpec pbeKeySpec = new PBEKeySpec(password.toCharArray(), toByte(salt), 50, 256);
SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("PBEWithSHA256And256BitAES-CBC-BC");
SecretKeySpec secretKey = new SecretKeySpec(keyFactory.generateSecret(pbeKeySpec).getEncoded(), "AES");
byte[] key = secretKey.getEncoded();
....

// setup AES cipher in CBC mode with PKCS7 padding
BlockCipherPadding padding = new PKCS7Padding();
BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(new CBCBlockCipher(new AESEngine()), padding);
cipher.reset();
cipher.init(false, params);
....
len += cipher.doFinal(buf, len);
```
<table>
<thead>
<tr>
<th>Feature</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation independence</td>
<td>good</td>
</tr>
<tr>
<td>Implementation simplicity</td>
<td>good</td>
</tr>
<tr>
<td>Implementation abstraction</td>
<td>good</td>
</tr>
<tr>
<td>Dynamic exchangeability and extensibility</td>
<td>mediocre</td>
</tr>
<tr>
<td>Manageability</td>
<td>mediocre</td>
</tr>
<tr>
<td>Portability</td>
<td>mediocre</td>
</tr>
<tr>
<td>Performance*</td>
<td>no</td>
</tr>
</tbody>
</table>

*compared to what is possible
What about PKI?
Agile Cryptography in PKI

Lack of Agility

- Todays PKI systems are not agile
  - No easy way to switch algorithms in **HSMs**
  - No easy way to switch algorithms in **software** stack
  - No easy way to switch algorithms in **certificates**
Agile Cryptography in PKI

Make it Agile

- **HSM and Software**
  - Make crypto updateable
  - Make crypto configurable
  - Make crypto manageable

- **Certificates**: ?
Agile Cryptography in PKI

- X.509 does support crypto agility
- Replacing hash or signature algorithm = replacing certificate
- Replacing certificates costly and time consuming
Summary
Crypto market has to change, we need crypto agility
- to be prepared for quantum computers
- to cope with the constant progress in cryptography
- to comply with local regulations
- to be prepared for future standards and challenges

Many open questions
- How can we achieve better data agility?
- How can we make IoT crypto agile?
- How can we make blockchain crypto agile?