Security primitives II

Secure channel

Secure storage

Secure envelope

PA193 – Secure coding

Petr Švenda
Partially based on slides prepared by Zdeněk Říha
Faculty of Informatics, Masaryk University, Brno, CZ

Centre for Research
Cryptography and Secu

Security primitives

- Secure channel
  - Communication
- Secure storage
  - Storage
- Secure envelope
  - Data protection
- Use standard, commonly used mechanisms
  - It is very difficult to create your own mechanisms that will be as secure as the standard ones

How to authenticate and communicate securely?

SECURE CHANNEL PROTOCOL

Secure channel

- Secure channel is a way of transferring data that is resistant to overhearing and tampering

Source: https://en.wikipedia.org/wiki/Secure_channel

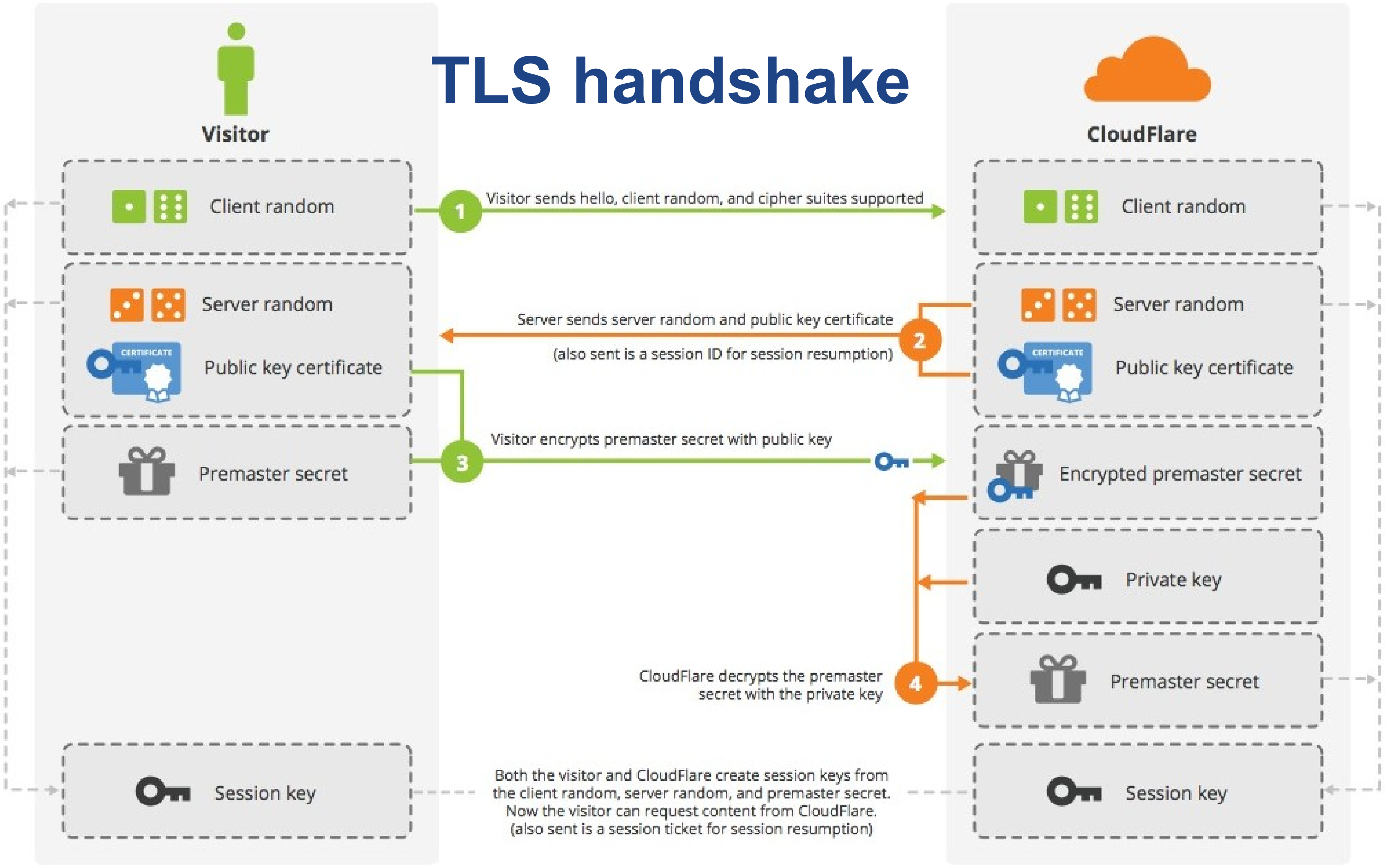
- Examples
  - Secure Messaging (smartcards)
    - ISO 7816-4
    - Open Platform / Global Platform
  - SSL/TLS
  - IPSEC
  - VPN

Transport Layer Security (TLS) Protocol

```
Client                                              Server

ClientHello                  -------->
                                               ServerHello
                                               Certificate
                             <--------     ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished                     -------->
                                        [ChangeCipherSpec]
                             <--------            Finished
Application Data             <------->    Application Data
```

Full TLS handshake (RFC 5246)

TLS handshake


Visitor
CloudFlare
Client random
Visitor sends hello, client random, and cipher suites supported
1
Client random
Server random
Public key certificate
Server sends server random and public key certificate
(also sent is a session ID for session resumption)
2
Server random
Public key certificate
Premaster secret
3
Visitor encrypts premaster secret with public key
Encrypted premaster secret
Private key
CloudFlare decrypts the premaster secret with the private key
4
Premaster secret
Session key
Both the visitor and CloudFlare create session keys from the client random, server random, and premaster secret. Now the visitor can request content from CloudFlare. (also sent is a session ticket for session resumption)
Session key

Credit: Cloudflare

Secure channels – questions to ask

- Integrity protection? Encryption? Authentication?
- What attacker model is assumed?
- One-side or mutual authentication?
- What kind of cryptography is used?
- What keys are required/pre-distributed?
- Additional trust hierarchy required?
- Is necessary to generate random numbers/keys?
- What if keys are compromised? Forward secrecy?

Case study SSL/TLS

- Let's look at the failure of SSL/TLS in more details
  - Read more at:
    - http://www.ieee-security.org/TC/SP2013/papers/4977a511.pdf
- Basic knowledge of SSL/TLS expected
  - Mandatory server authentication
  - Optional client authentication
  - Authentication based on X.509 certs and private key
  - PKI infrastructure to validate certs needed
  - Confidentiality and integrity provided
  - Non-repudiation not provided

Weaknesses in Crypto Primitives

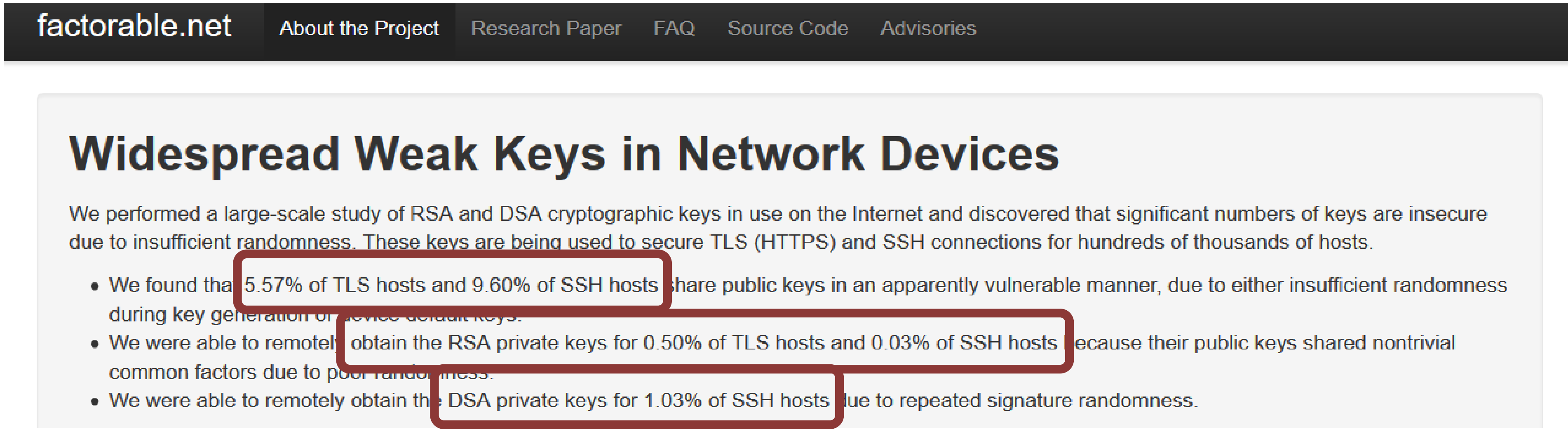
- SSL/TLS started with 40/56 bit symmetric keys
  - DES, RC2, RC4
  - Due to US export regulation
- Slow changes, backward compatibility maintained
- Still possible to see certs with unsecure parameters
  - Based on RSA-512 (factorable today!)
  - (Google was using RSA-1024 until Nov 2013)
  - Based on MD5 (collision attack on certs demonstrated)
- Google/Chrome now active in pushing stronger security
  - Certificate Transparency, removed rogue CAs, SHA-1 phase out, post-quantum cipher suite CECPQ1 …

PRNG problems

- Netscape browser prior 1.22 relied on weak PRNG for SSL
- Debian problems with entropy gathering
  - Predictable OpenSSL keys
- Insufficient entropy during device startup
  - Factorable TLS keys
- …

Weak TLS keys (2012)

- Internet wide scans, scans.io/, censys.io/
- Attempts to factorize fraction of keys
  - Shared prime between two or more keys (GCD attack), insufficient entropy during device start, repeated randomness in DSA signatures…



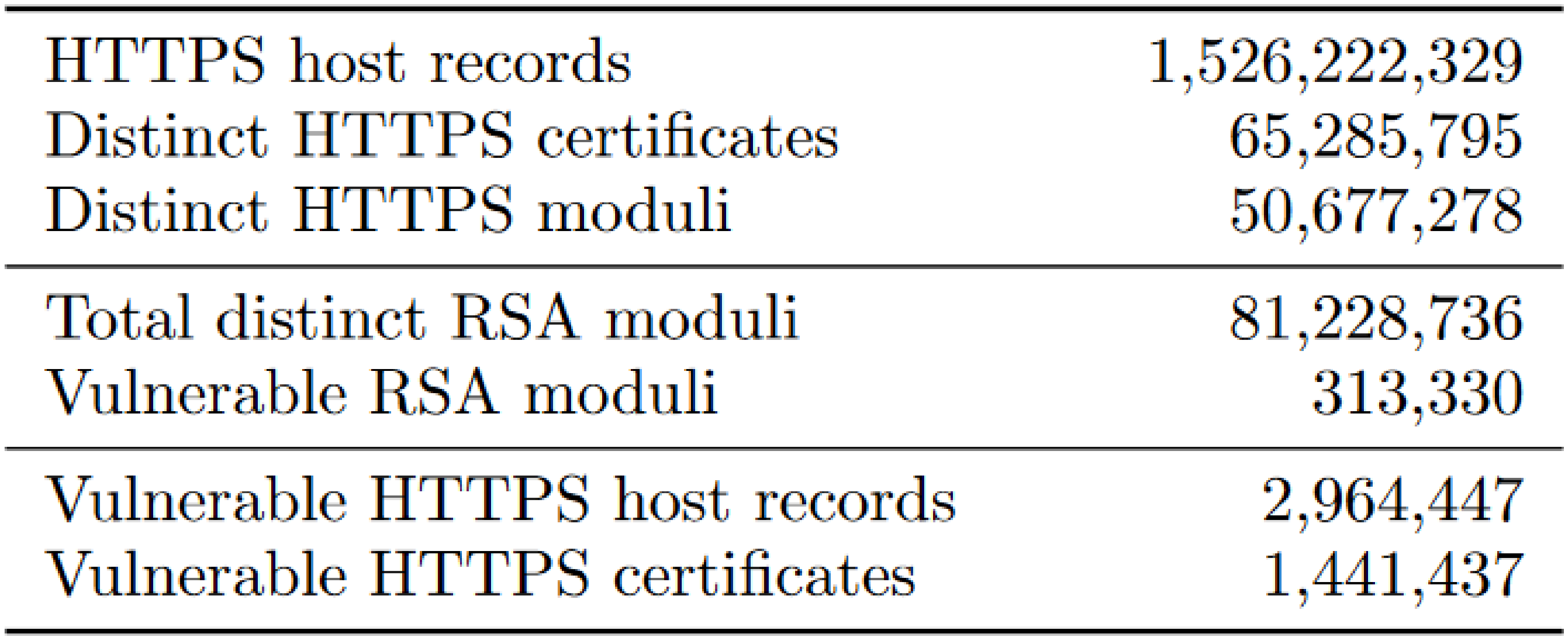
Widespread Weak Keys in Network Devices

We performed a large-scale study of RSA and DSA cryptographic keys in use on the Internet and discovered that significant numbers of keys are insecure due to insufficient randomness. These keys are being used to secure TLS (HTTPS) and SSH connections for hundreds of thousands of hosts.

- We found that 5.57% of TLS hosts and 9.60% of SSH hosts share public keys in an apparently vulnerable manner, due to either insufficient randomness during key generation or device default keys.
- We were able to remotely obtain the RSA private keys for 0.50% of TLS hosts and 0.03% of SSH hosts because their public keys shared nontrivial common factors due to poor randomness.
- We were able to remotely obtain the DSA private keys for 1.03% of SSH hosts due to repeated signature randomness.

Weak TLS keys remain widespread (2016)

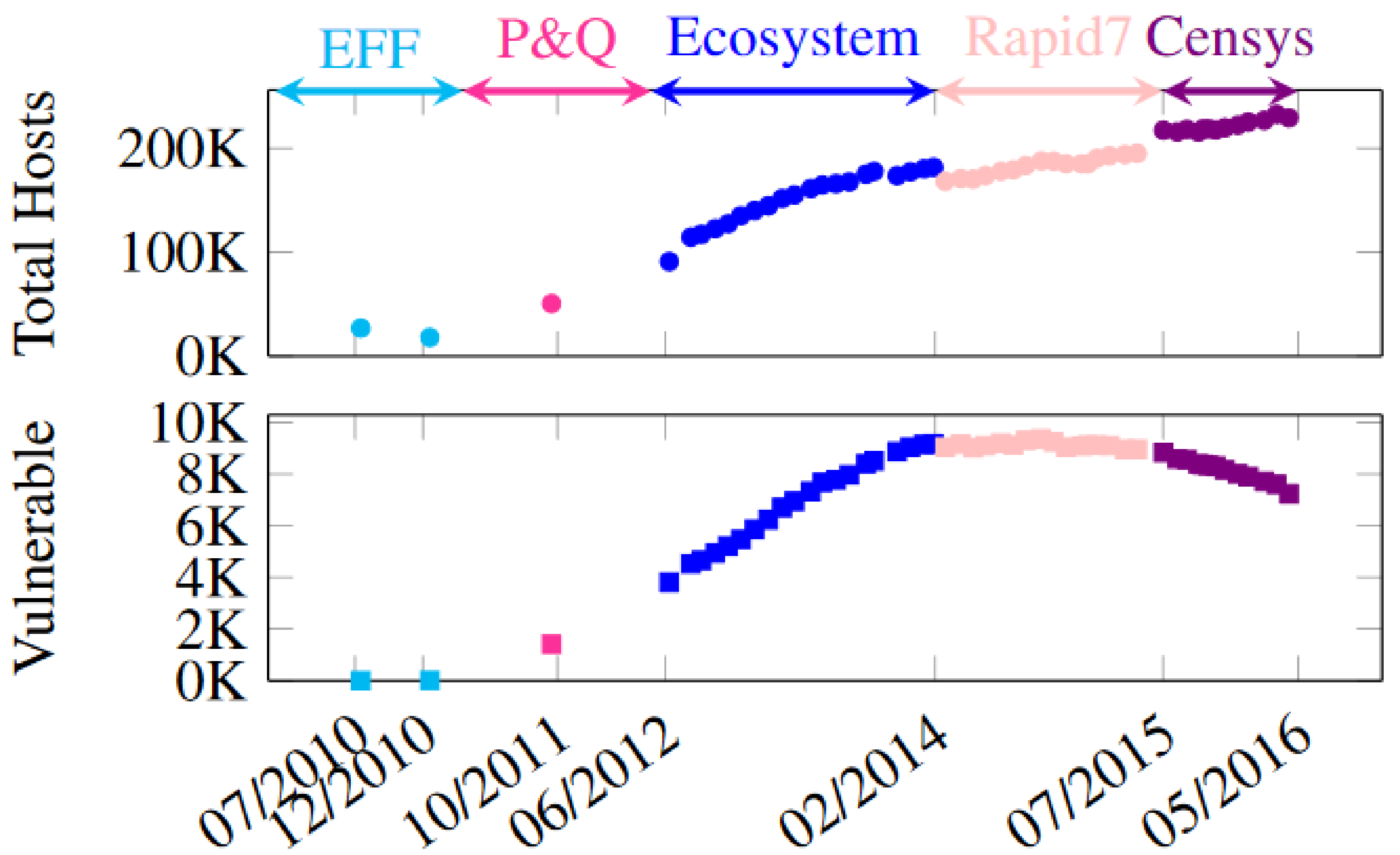
- GCD factorization of TLS keys between 2010-2016



HTTPS host records	1,526,222,329
Distinct HTTPS certificates	65,285,795
Distinct HTTPS moduli	50,677,278
Total distinct RSA moduli	81,228,736
Vulnerable RSA moduli	313,330
Vulnerable HTTPS host records	2,964,447
Vulnerable HTTPS certificates	1,441,437

M. Hastings et. al.: https://dl.acm.org/citation.cfm?id=2987486

Example: weak TLS keys for Cisco devices



M. Hastings et. al.: https://dl.acm.org/citation.cfm?id=2987486

Developer defences against weak RNG

- Use statistical randomness test suites
  - STS NIST, Dieharder, TestU01
  - Usually applied during integration testing
- Add simple runtime checks (self-test, number of 1&0…)
  - Can be performed in production code, required by FIPS140-2
- Don’t generate keys during boot (embedded devices)
  - Not enough entropy present yet
- Use more sources of entropy (Intel’s RNDRAND, TPM)
  - Should be already utilized by kernel rng, but not always
  - Add additional entropy, do NOT replace existing!
- Inject additional entropy continuously
  - Significantly harder to reconstruct, recovery from state compromise

Remote timing attacks

- Against SSL servers using optimized RSA decryption based on OpenSSL
  - Server response correlated with bits of private key
  - optimized decryption was default in OpenSSL prior 0.9.7b
- The long term secret of the server was leaking during the SSL/TLS handshake
- Solution 1: decrease measurement precision
  - add noise, limit granularity… - generally difficult
- Solution 2: constant-time implementations
  - E.g., OpenSSL after fix, NaCL library
- More in PV204 Security Technologies (Spring)

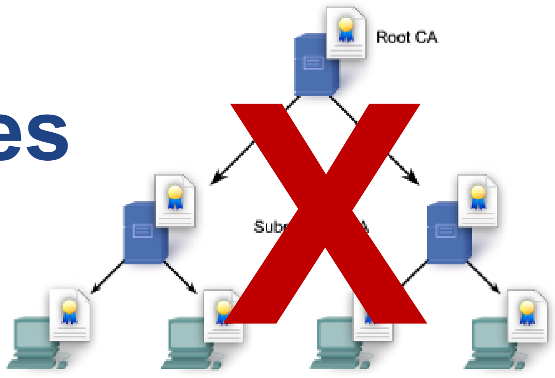
Protocol attacks

- Ciphersuite downgrade
  - In SSL 2.0, downgrade to 40bit RC4
  - Padding Oracle On Downgraded Legacy Encryption (POODLE)
- SSL Version downgrade
  - If clients misinterpret higher version error and try to continue with a lower protocol version
- Cross-protocol attacks (DROWN)
  - Server supporting SSL 2.0 as oracle to decrypt TLS 1.2
- Renegotiation attack
  - Renegotiate security related parameters

Trust model - X.509 Certificates

- Hostname Validation
  - Do not skip the hostname validation
- Study: Analysis of Android SSL (in)security
  - over 1000 out of 13500 popular free Android applications do not validate the hostname
  - http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf
- Android SSL guidelines
  - https://developer.android.com/training/articles/security-ssl.html
- nogotofail project: https://github.com/google/nogotofail
  - MitM tool for correct SSL use

Trust model - X.509 Certificates



Hierarchical CA Topology

- Anchoring trust
  - X.509 original idea: single world-wide CA
  - Reality: multiple non-trusting CAs
  - Web browsers include +-150 trust points from +-50 organizations
- What are problems with many CAs?
  - CA compromise or negligence (DigiNotar,CNNIC,TURKTRUST…)
  - The power of governments over CAs
  - Transitivity of trust (basicConstrains – CA:TRUE)
    - This flag must be checked otherwise anybody could be validated as any web site (MS CryptoAPI & Apple iOS did not)

https://sclabs.blogspot.cz/2012/10/ccna-security-chapter-7-cryptographic.html

Trust model - X.509 Certificates

- Parsing attacks
  - Binary 0 in CN => google.com0evil.com validated as google.com
- Revocation
  - How to authenticate revocation request?
  - Blocking the revoked (stolen/incorrect) certificates
    - Online Certificate Status Protocol (OCSP)
    - Certificate Revocation List (CRL)
  - Problem: Is OCSP/CRL provided?
  - Problem: Most clients will silently ignore OCSP timeout
    - MitM attacker with server’s private key blocks OCSP (on cable)
  - OCSP stapling – time-stamped OCSP during initial TLS handshake

HTTP vs. HTTPS

- Stripping TLS
  - a man-in-the-middle attack
  - relay HTTPS pages over HTTP
  - Victim $\leftarrow$ HTTP $\rightarrow$ Attacker $\leftarrow$ HTTPS $\rightarrow$ Server
- The SSLstrip tool
  - Potentially running by ISP, gateway, router…
  - https://github.com/moxie0/sslstrip
- HTTP Strict Transport Security policy as protection

HTTP Strict Transport Security (HSTS)

- Web security policy mechanism
  - Web server declares that clients should interact only via secure HTTPS connections
  - The policy is communicated via a HTTP response header called "`Strict-Transport-Security`"
- But how to protect initial HTTP header?
  - Preloaded list of known HSTS servers in browser
    - Now supported by Chrome, Firefox, Edge…
    - Non-browser software libraries might be lacking

Implementation notes

1. Select proper secure channel for your requirements
2. Select proper library, follow security advisories
3. Understand what checks are performed by library
4. Understand your key management
5. Understand your trust hierarchy
6. Write positive and negative tests (invalid cert…)
7. Don’t forget to (verify) check revocations
8. Make sure your RNG generator is correct
9. Be prepared for future updates
   – broken/new ciphers, root of trust, version of library…

H('Password') →

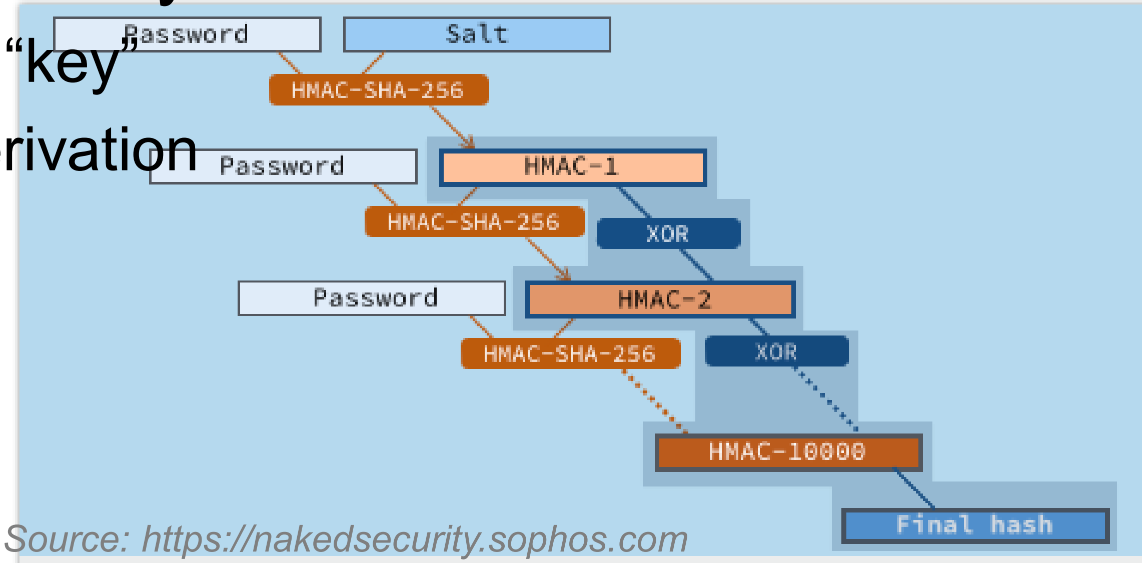
METHODS OF DERIVATION OF SECRETS FROM PASSWORD

Problems when password used as a key

- Passwords are usually shorter / longer than key
- If password as a key => low number of distinct keys
- Password does not contain same amount of entropy as binary key (only printable characters…)
- K = SHA-2("password")
  - Same passwords from multiple users => same key
  - Large pre-computed "rainbow" tables allow for quick check
  - Solved by addition of random (potentially public) salt
    - K = SHA-2(pass | salt)
- Dictionary-based brute-force still possible

Derivation of secrets from password

- PBKDF2 function, widely used
  - Password is HMAC "key"
  - Iterations to slow derivation
  - Salt added



Source: https://nakedsecurity.sophos.com

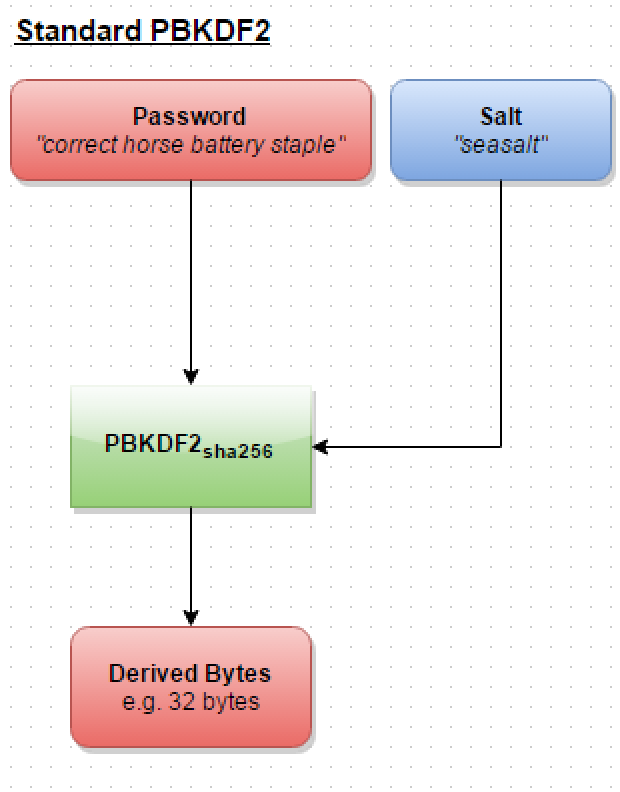
PBKDF2 using XOR to combine 10,000 successive HMAC-SHA-256 outputs into a final has

- Problem with custom-build hardware (GPU, ASIC)
  - Repeated iterations not enough to prevent bruteforce
  - (or would be too slow on standard CPU – user experience)

scrypt – memory hard function

- Design as a protection against cracking hardware (usable against PBKDF2)
  - GPU, FPGA, ASICs…
  - https://github.com/wg/scrypt/blob/master/src/main/java/com/lambdaworks/crypto/SCrypt.java
- Memory-hard function
  - Force computation to hold r (parameter) blocks in memory
  - Uses PBKDF2 as outer interface
- Improved version: NeoScrypt (uses full Salsa20)

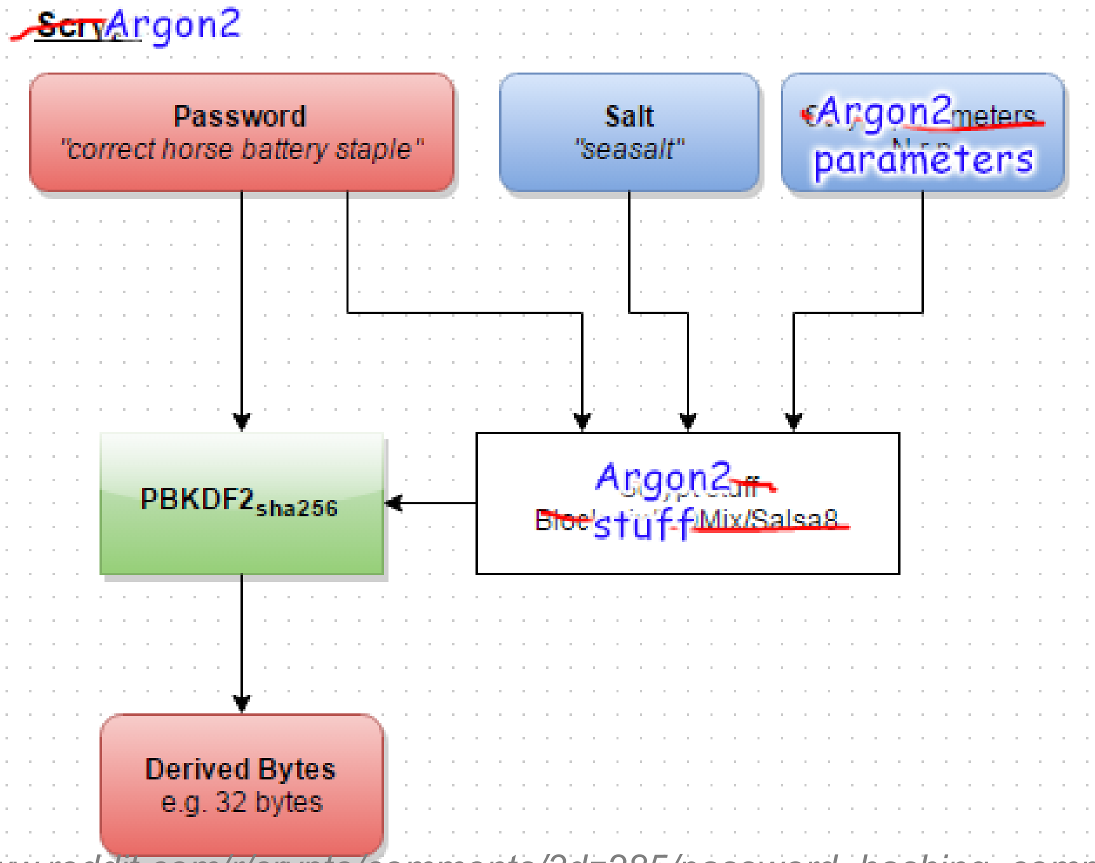
Reuse of external PBKDF2 structure



https://www.reddit.com/r/crypto/comments/3dz285/password_hashing_competition_phc_has_selected/

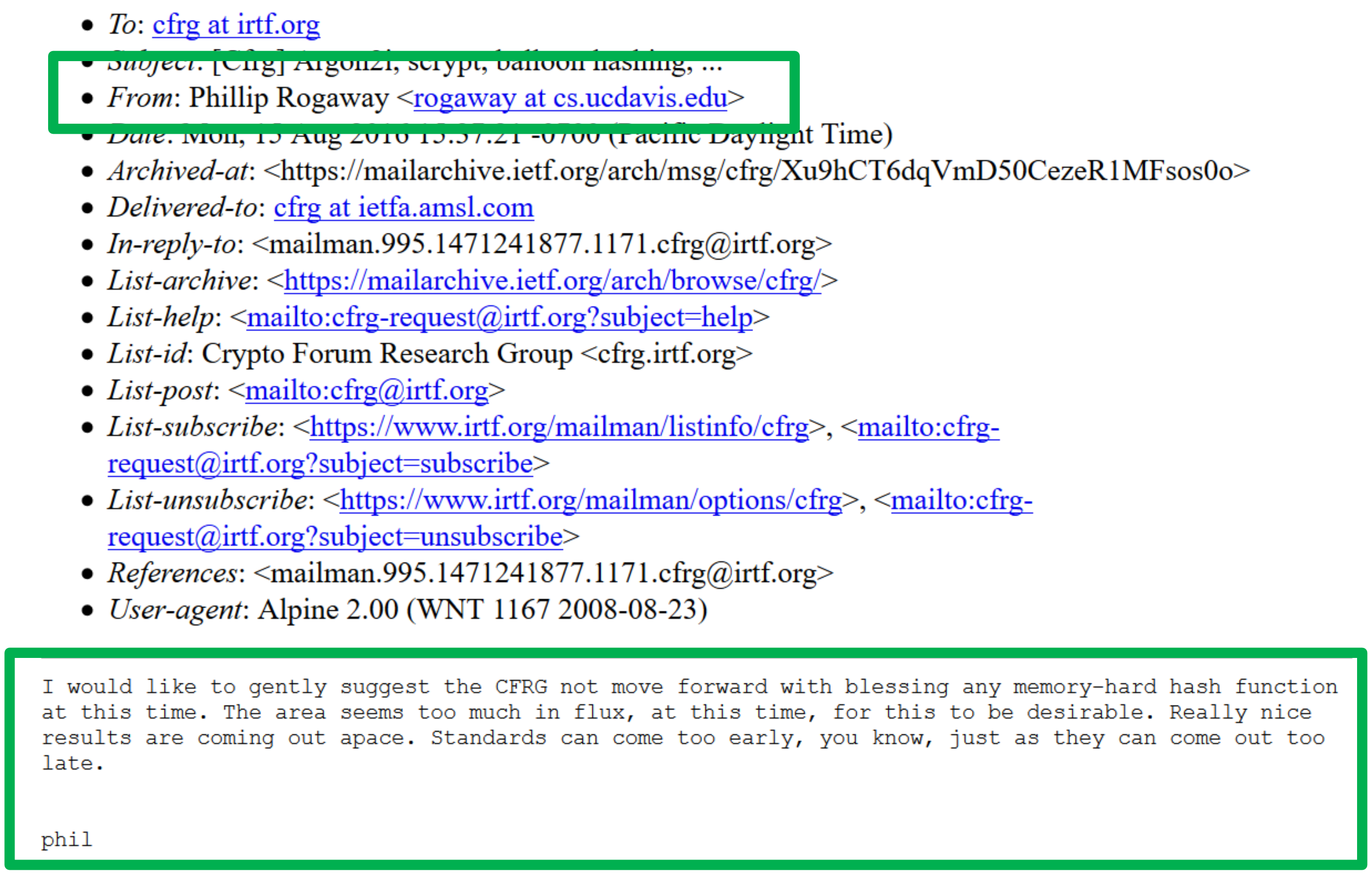
Argon2

- Password hashing competition (PHC) winner, 2013



https://www.reddit.com/r/crypto/comments/3dz285/password_hashing_competition_phc_has_selected/

Problem solved?



- *To*: cfrg at irtf.org
- *Subject*: [Cfrg] Argon2i, scrypt, balloon hashing, ...
- *From*: Phillip Rogaway <rogaway at cs.ucdavis.edu>
- *Date*: Mon, 15 Aug 2016 15:37:21 -0700 (Pacific Daylight Time)
- *Archived-at*: <https://mailarchive.ietf.org/arch/msg/cfrg/Xu9hCT6dqVmD50CezeR1MFsos0o>
- *Delivered-to*: cfrg at ietfa.amsl.com
- *In-reply-to*: <mailman.995.1471241877.1171.cfrg@irtf.org>
- *List-archive*: <https://mailarchive.ietf.org/arch/browse/cfrg/>
- *List-help*: <mailto:cfrg-request@irtf.org?subject=help>
- *List-id*: Crypto Forum Research Group <cfrg.irtf.org>
- *List-post*: <mailto:cfrg@irtf.org>
- *List-subscribe*: <https://www.irtf.org/mailman/listinfo/cfrg>, <mailto:cfrg-request@irtf.org?subject=subscribe>
- *List-unsubscribe*: <https://www.irtf.org/mailman/options/cfrg>, <mailto:cfrg-request@irtf.org?subject=unsubscribe>
- *References*: <mailman.995.1471241877.1171.cfrg@irtf.org>
- *User-agent*: Alpine 2.00 (WNT 1167 2008-08-23)

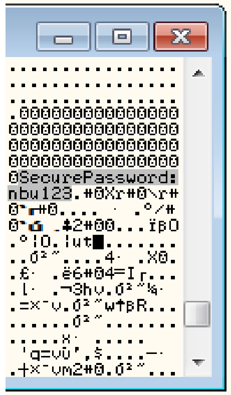
```
I would like to gently suggest the CFRG not move forward with blessing any memory-hard hash function
at this time. The area seems too much in flux, at this time, for this to be desirable. Really nice
results are coming out apace. Standards can come too early, you know, just as they can come out too
late.


phil
```

https://www.ietf.org/mail-archive/web/cfrg/current/msg08439.html

SECURE STORAGE

Secure storage: how to keep secrets secret

- Secret data
  - Symmetric encryption keys, asymmetric private keys
  - Passwords…
- Storing secrets in software-only
  - Completely securely: **IMPOSSIBLE**
    - Reverse engineering of binaries
    - Debugging, paging memory to files
    - Malicious administrators…
- Storing secrets in HW (HSMs, smartcards…)
  - Option 1: Protection of secret data before use
    - Potential compromise during use
  - Option 2: Protection of secret data also during use
    - Problem with access control (authentication of use of secret)



```
................
................
................
.000000000000000
0000000000000000
0000000000000000
0000000000000000
0SecurePassword:
nbu123.#0Xr#0\r#
0˜r#0....  ·  .°/#
0˜ô  _♣2#00...ïβO
.°¦O.¦ut█.......
..0²˝.....4·  .X0.
.£·  .ë6#04=I┌...
.l·  .¬3hv.0²˝¼·
.=x¯v.0²˝w†βR...
.......0²˝.......
......x·  .....
 'q=vÛ',§.....—·
.+x¯vm2#0.0²˝...
```

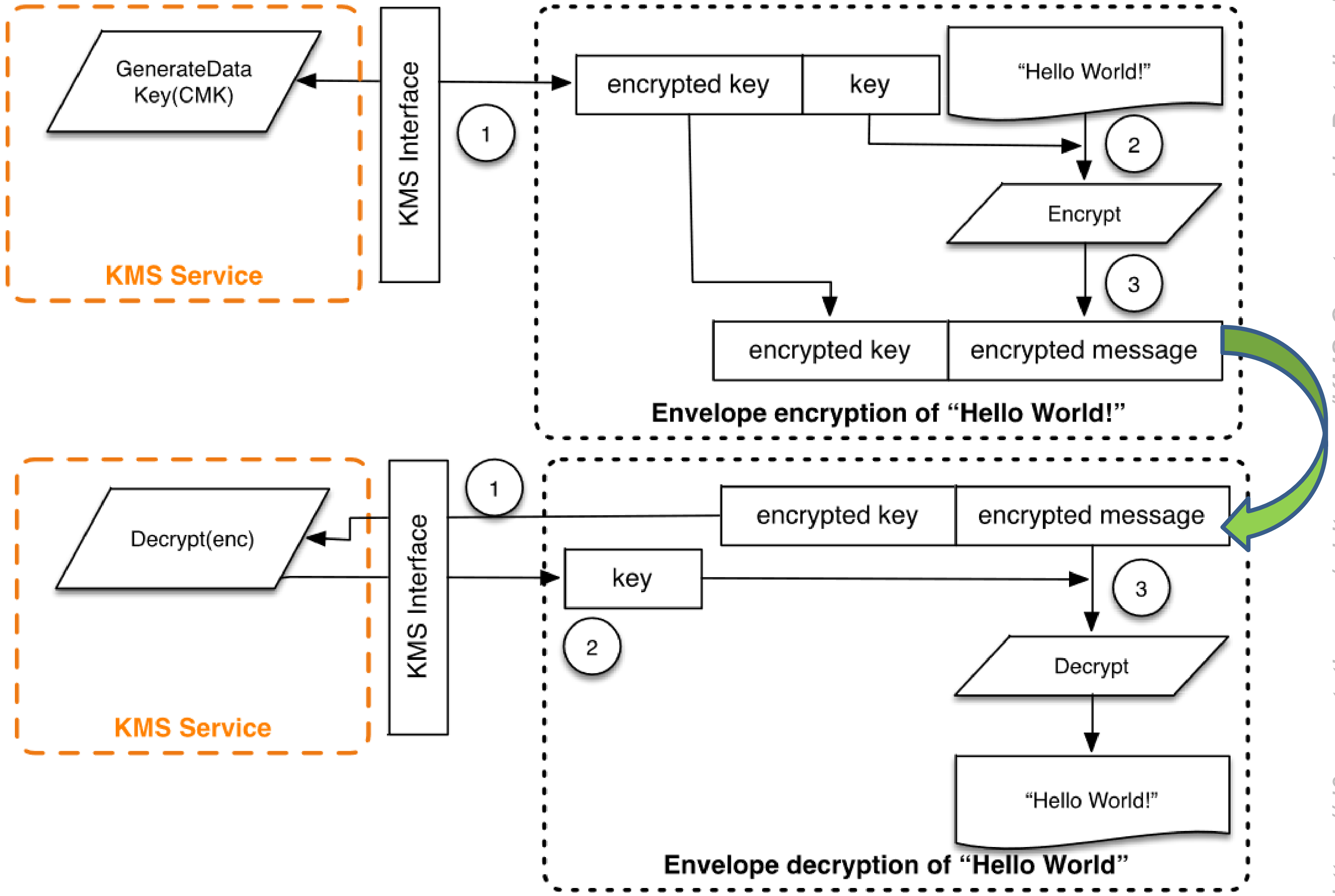
Design / implementation notes

1. Understand your use case scenario
   - Who should be able to access protected data? (system/user/rec.agent)
   - Should data be accessible also on other devices? (device key/sealing)
   - Should data be accessible even when device is locked?
   - Is user required to insert password before access to data?
2. Select proper system API, application or library
   - Your target platform/OS, proper OS layer (kernel vs. user-mode)
   - OS provided vs. independent secure storage (e.g., KeePass)
   - Protection of file vs. data blob vs. password
3. Understand security model and design
   - Who can access files during recovery?
   - What if device is lost/stolen (attacker with physical access)?

Secure storage in OSes

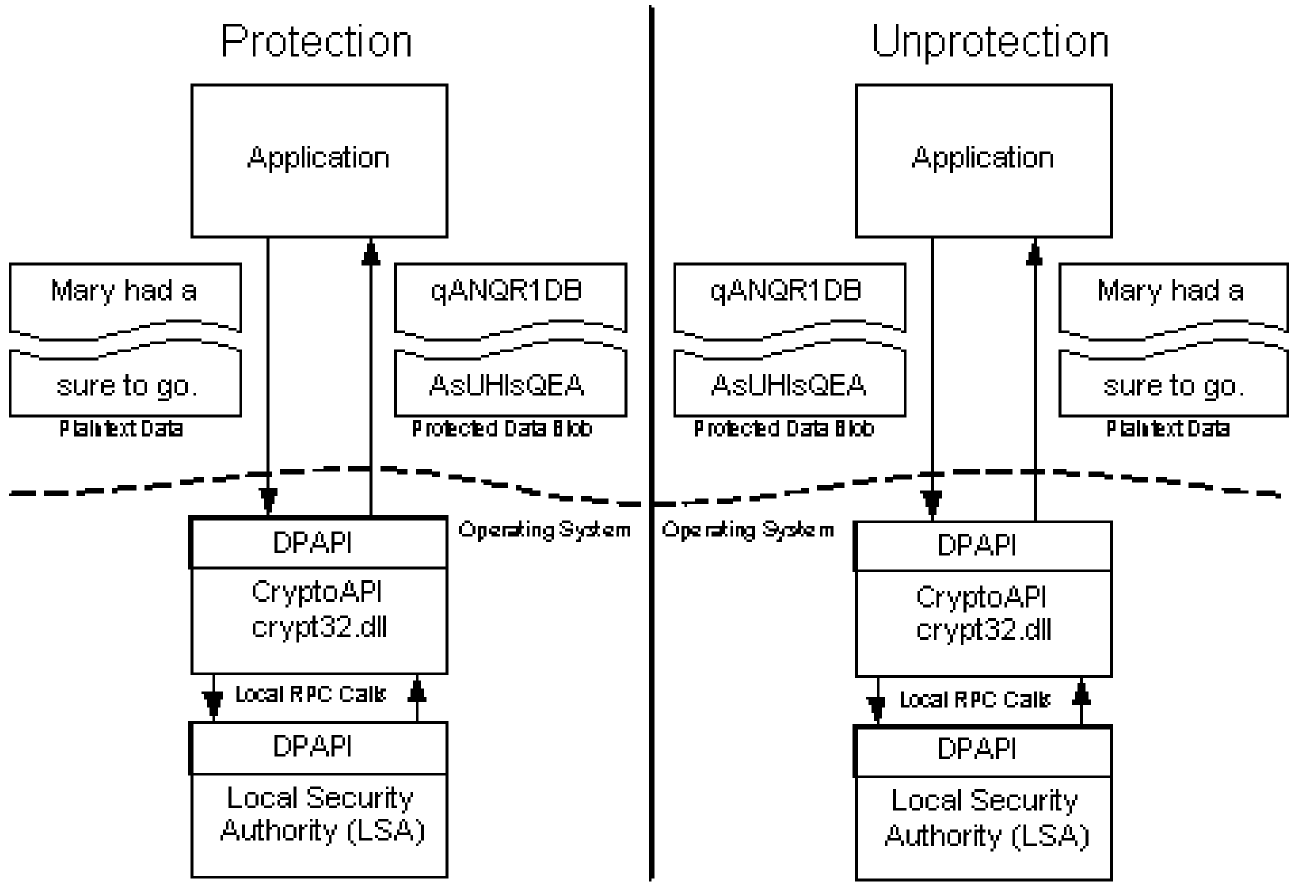
- MS Windows: **Data Protection API (DPAPI)**
  - CryptProtectData(), CryptUnprotectData()
- Apple OS X, iOS: **Keychain Services API**
  - SecKeychainAddGenericPassword(), SecKeychainFindGenericPassword()
  - FS: NSFileProtectionNone, NSFileProtectionComplete
- Linux kernel: **keyutils**
  - add_key(), request_key()
- Linux GNOME: **gnome-keyring-manager**
- Linux KDE: **kwallet**

Example: AWS Key Management Service



https://d0.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf

Example MS DPAPI



https://msdn.microsoft.com/en-us/library/ms995355.aspx

Protecting secrets in Windows

- Data Protection API (DPAPI)
  - CryptProtectData(), CryptUnprotectData()
- Data available to user
  - Bound with user account, available on multiple machines but not on other accounts
- Data available to machine
  - Available to any user at the machine, not available at other machines
  - Use CRYPTPROTECT_LOCAL_MACHINE flag

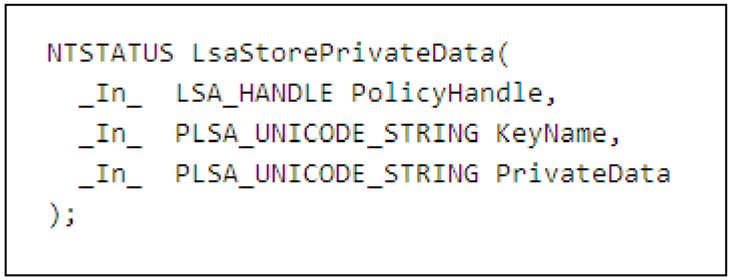
Protecting secrets on Windows

- DAPI does not provide storage of protected blobs
  – only encryption/decryption
- You have to manage storage yourself
  – Be careful to protect the encrypted data with correct ACLs in files/registry

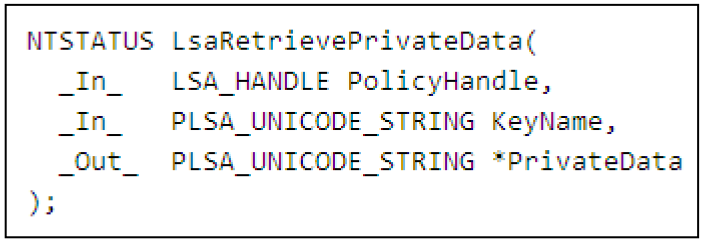
- Any application running on the USER can decrypt the secrets!
- If you do not like this, use `pOptionalEntropy` field
  – To protect your secrets with another secret ☺

LSA interface (old stuff)

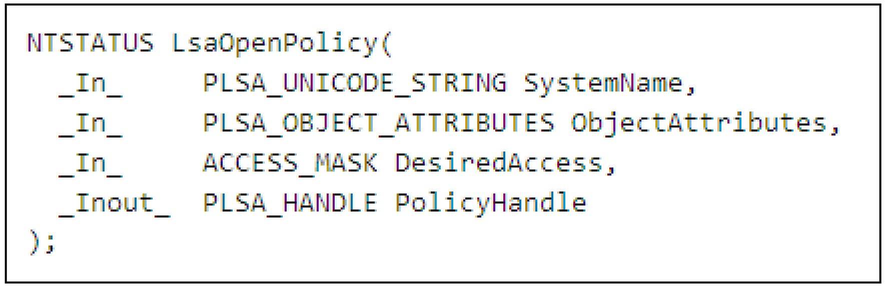
- “The Local Security Authority (LSA) is a protected subsystem of Windows that maintains information about all aspects of local security on a system.”
- LSA secrets:



```
NTSTATUS LsaStorePrivateData(
  _In_  LSA_HANDLE PolicyHandle,
  _In_  PLSA_UNICODE_STRING KeyName,
  _In_  PLSA_UNICODE_STRING PrivateData
);
```



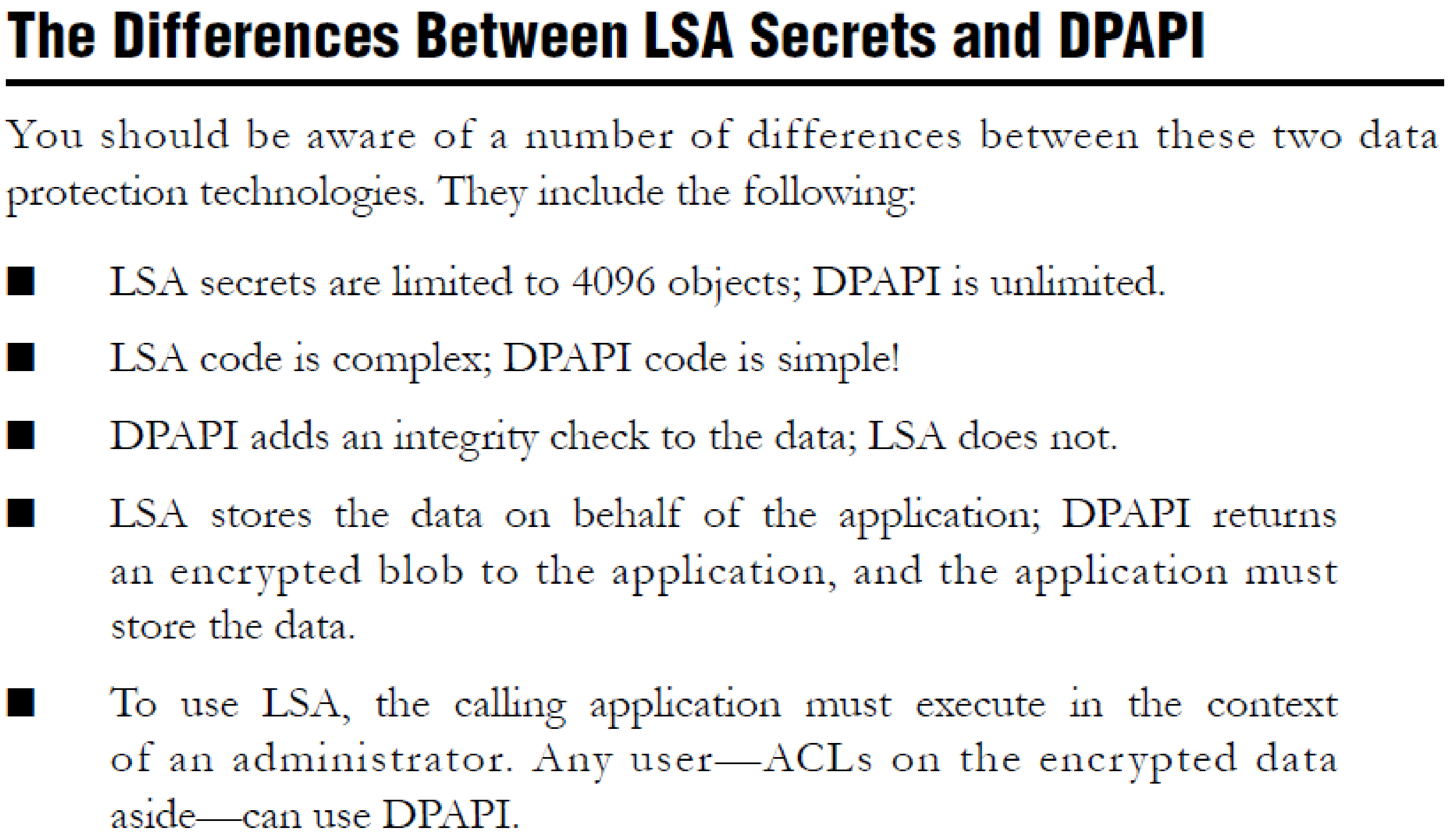
```
NTSTATUS LsaRetrievePrivateData(
  _In_   LSA_HANDLE PolicyHandle,
  _In_   PLSA_UNICODE_STRING KeyName,
  _Out_  PLSA_UNICODE_STRING *PrivateData
);
```



```
NTSTATUS LsaOpenPolicy(
  _In_     PLSA_UNICODE_STRING SystemName,
  _In_     PLSA_OBJECT_ATTRIBUTES ObjectAttributes,
  _In_     ACCESS_MASK DesiredAccess,
  _Inout_  PLSA_HANDLE PolicyHandle
);
```

LSA interface – LSA secrets

- LSA secrets:
  - Local data
    - Can be read only at the machine storing data (L$)
  - Global data
    - Created on domain controller and replicated (G$)
  - Machine data
    - Can be accessed only by OS (M$)
  - Private data
    - Can be used by your application



The Differences Between LSA Secrets and DPAPI

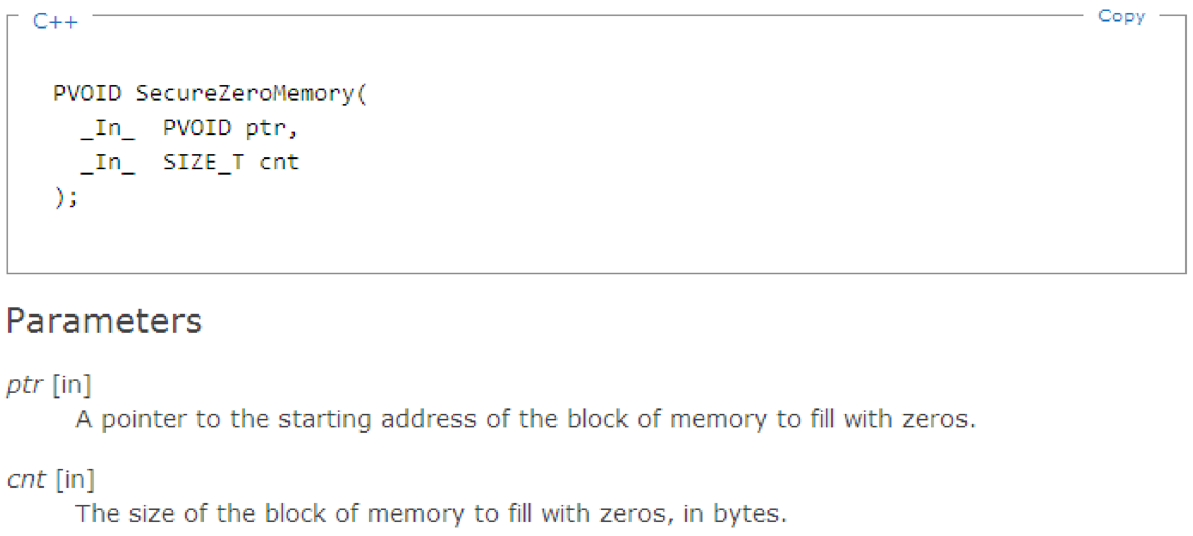
You should be aware of a number of differences between these two data protection technologies. They include the following:

- LSA secrets are limited to 4096 objects; DPAPI is unlimited.
- LSA code is complex; DPAPI code is simple!
- DPAPI adds an integrity check to the data; LSA does not.
- LSA stores the data on behalf of the application; DPAPI returns an encrypted blob to the application, and the application must store the data.
- To use LSA, the calling application must execute in the context of an administrator. Any user—ACLs on the encrypted data aside—can use DPAPI.

Source: Writing secure code, 2nd edition

Managing secrets in memory

- ZeroMemory()
  - Macro using memset
- Compiler optimizations can remove the call of the function!!!
- Use SecureZeroMemory() instead



```
PVOID SecureZeroMemory(
  _In_  PVOID ptr,
  _In_  SIZE_T cnt
);
```

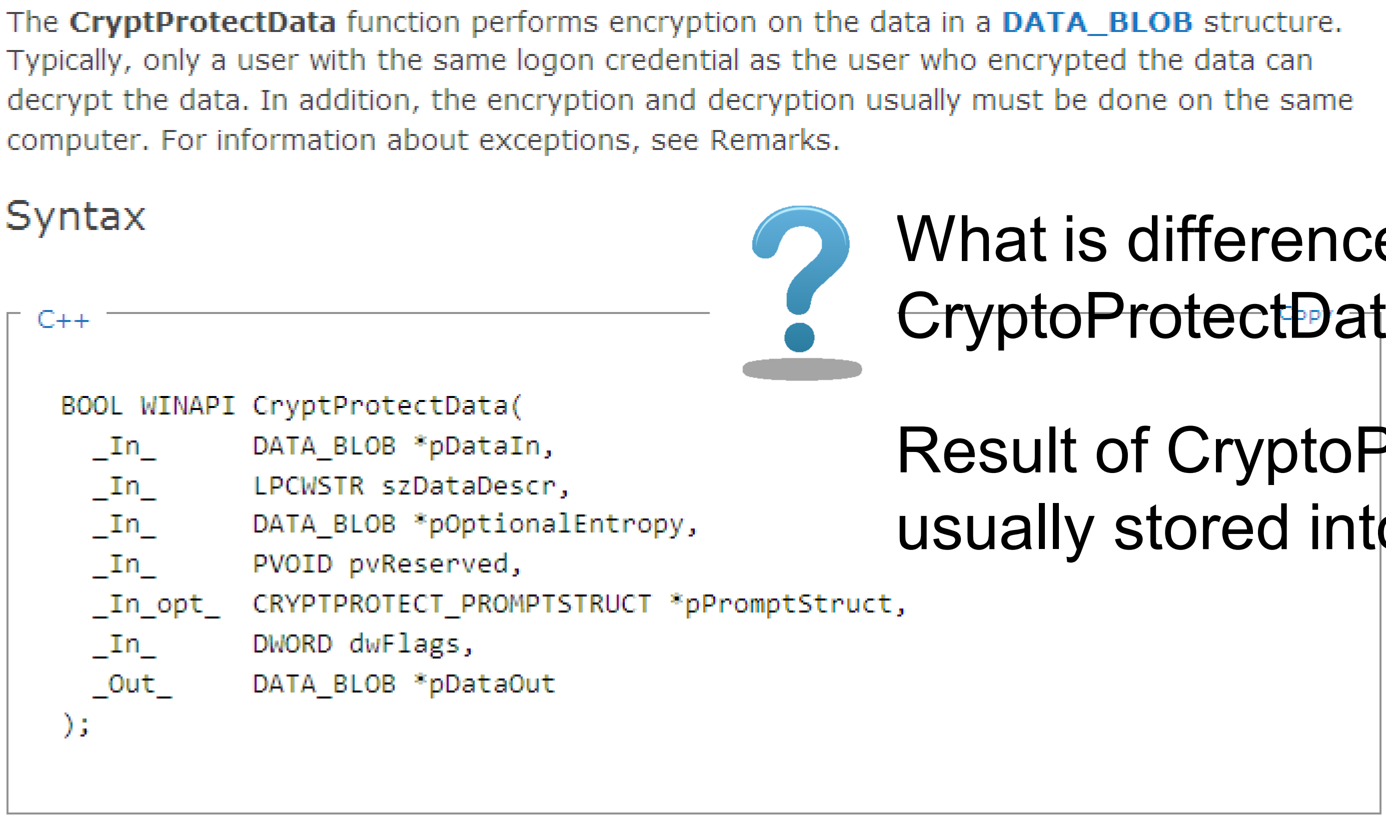
Parameters

ptr [in]
  A pointer to the starting address of the block of memory to fill with zeros.

cnt [in]
  The size of the block of memory to fill with zeros, in bytes.

Managing secrets in Memory

- CryptProtectMemory() and CryptUnprotectMemory()



The **CryptProtectData** function performs encryption on the data in a **DATA_BLOB** structure. Typically, only a user with the same logon credential as the user who encrypted the data can decrypt the data. In addition, the encryption and decryption usually must be done on the same computer. For information about exceptions, see Remarks.

Syntax

C++

```
BOOL WINAPI CryptProtectData(
  _In_      DATA_BLOB *pDataIn,
  _In_      LPCWSTR szDataDescr,
  _In_      DATA_BLOB *pOptionalEntropy,
  _In_      PVOID pvReserved,
  _In_opt_  CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
  _In_      DWORD dwFlags,
  _Out_     DATA_BLOB *pDataOut
);
```

What is difference from CryptoProtectData (DAPI)?

Result of CryptoProtectData usually stored into file.

Source: MSDN

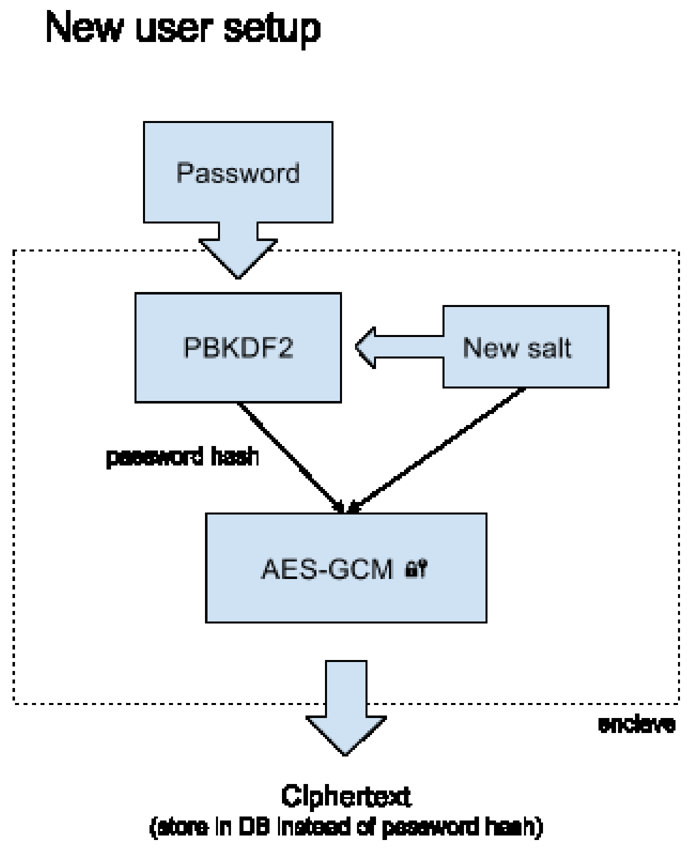
Locking Memory to Prevent Paging

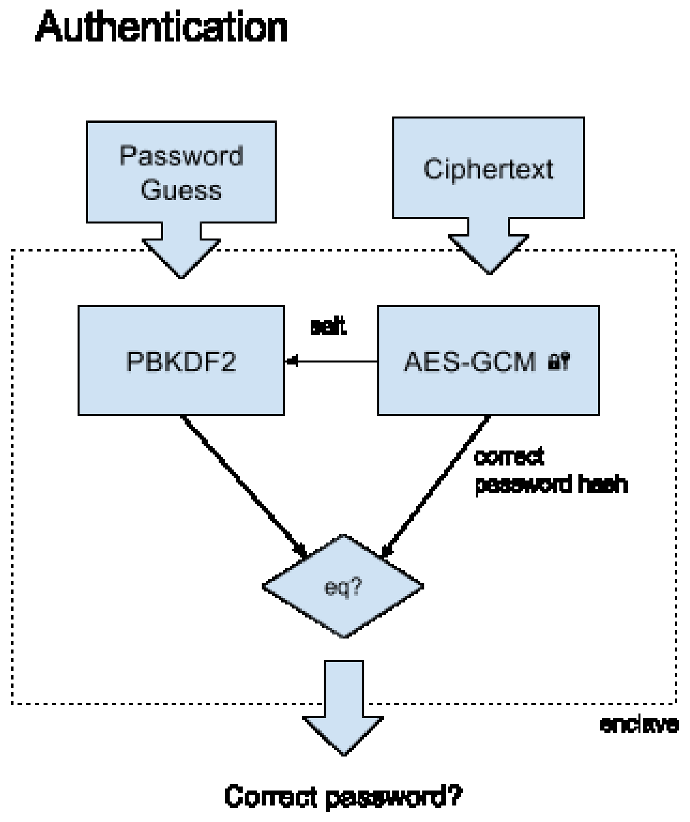
- To keep your sensitive data in RAM memory only
  - not in a paging file (security and performance implications)
- Lock memory before storing the secrets
  - VirtualLock()
  - AllocateUserPhysicalPages()
- Does not prevent dumping memory to disk when hibernating (or crash dump file)
- Does not prevent a debugger to read the memory

Intel’s SGX : Security enclave

- Intel’s Software Guard Extension (SGX)
  - New set of CPU instructions intended for future cloud server CPUs
  - EGETKEY, EREPORT
- Protection against privileged attacker
  - Server admin with physical access, privileged malware
- Application requests private region of code and data
  - Security enclave (4KB for heap, stack, code)
  - Encrypted enclave is stored in main RAM memory, decrypted only inside CPU
  - Access from outside enclave is prevented on CPU level
  - Code for enclave is distributed as part of application

SGX hardened password verification





https://jbp.io/2016/01/17/using-sgx-to-hash-passwords/

SECURE ENVELOPE

Protect data

- For secret/private keys use
  - PKCS#8
  - PKCS#12 (pfx)
- For digital signatures use
  - Cryptographic message Syntax (CMS, PKCS#7)
  - Secure email
    - S/MIME
      - Based on X.509 certificates
      - Transparent vs. opaque signing
    - PGP

PKCS#8

- Format for storing private key
- Independent on private key algorithm
- Key can be encrypted
- File suffix “.pkcs8”

```
PrivateKeyInfo ::= SEQUENCE {
   version Version,
   privateKeyAlgorithm AlgorithmIdentifier {{PrivateKeyAlgorithms}},
   privateKey PrivateKey,
   attributes [0] Attributes OPTIONAL }
```

PKCS#12

- Collection of cryptographic objects
- Privacy/confidentiality
  - Public key privacy mode: encrypted by a public key
  - Password privacy mode: encrypted by a symmetric key derived from username and password
- Integrity modes
  - Public key integrity mode: digital signature
  - Password protection mode: MAC based on password
- File suffix “.p12”, “.pfx”.

PKCS#12

- “SafeContents” is made up of “SafeBags”
- SafeBag types:
  - KeyBag: PKCS#8 private key
  - PKCS8ShroudedKeyBag: private key, which has been shrouded (encrypted) in accordance with PKCS #8
  - CertBag: certificate (X.509, SDSI – Simple Distributed Security Infrastructure)
  - CRLBag: CRL (X.509)
  - SecretBag: any other secret of a user

PKCS#7 / CMS

- Encapsulated content
- Provides for low-level message functions
- Content types:
  - Data (any plaintext)
  - Signed Data (digital signature based on X.509 certs)
  - Enveloped Data (encrypted data)
    - key transport: symmetric key encrypted by the recipient's pub key
    - key agreement: pairwise symmetric key created using the recipient's public key and the sender's private key
    - symmetric key-encryption keys: using a previously distributed key
    - passwords: key is derived from a password
  - Authenticated data (MAC + MAC key)
- OpenSSL, Microsoft CryptMsgXXX() functions…

Conclusions

- Important to understand what you want to achieve

1. Protection of stream of data in transport
   - Secure channel
2. Binding of data to device/user
   - Data protection API, secure storage, keyrings
3. Protection of data in memory
   - In memory encryption

- Key management is critical (as usual ☺)
  - Where are keys for establishment/storage stored?
- Secure hardware helps (combination with kernel)

Questions ?