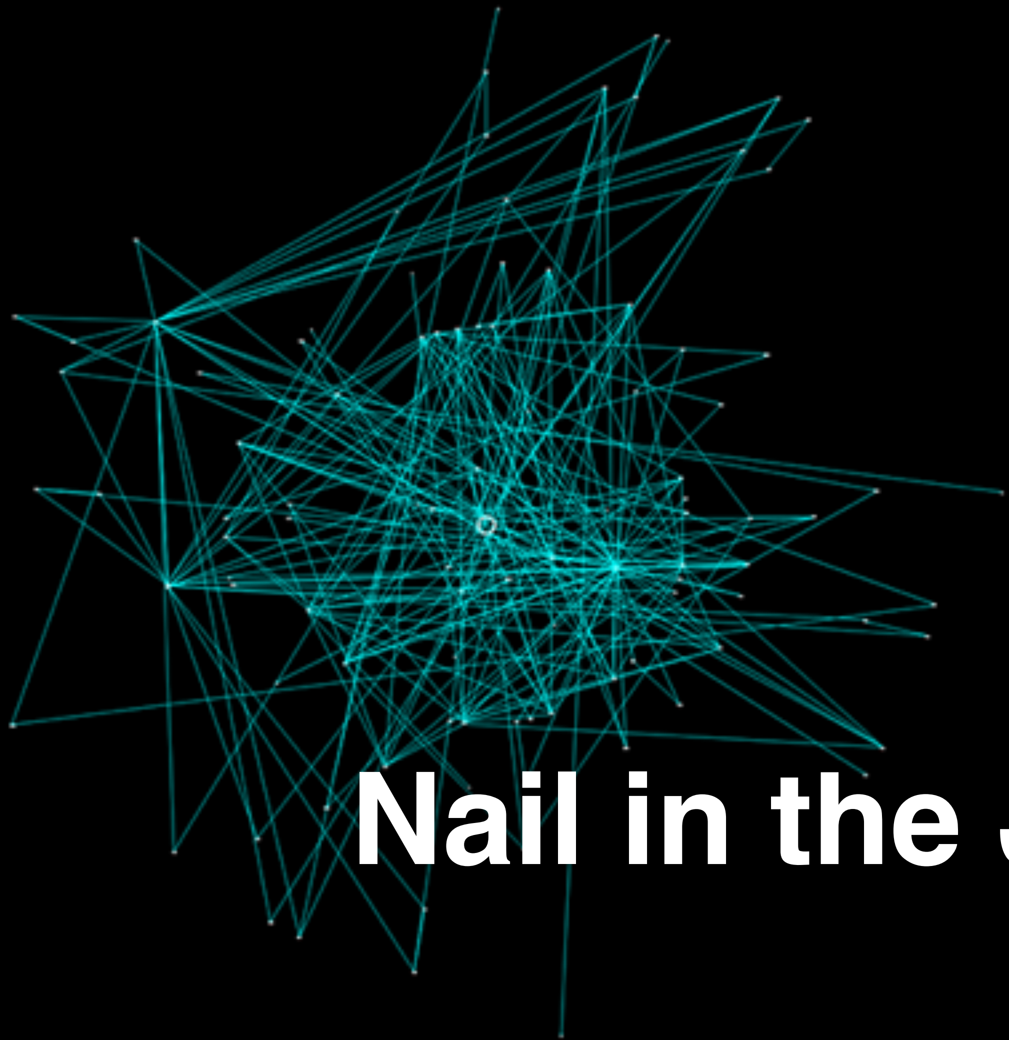


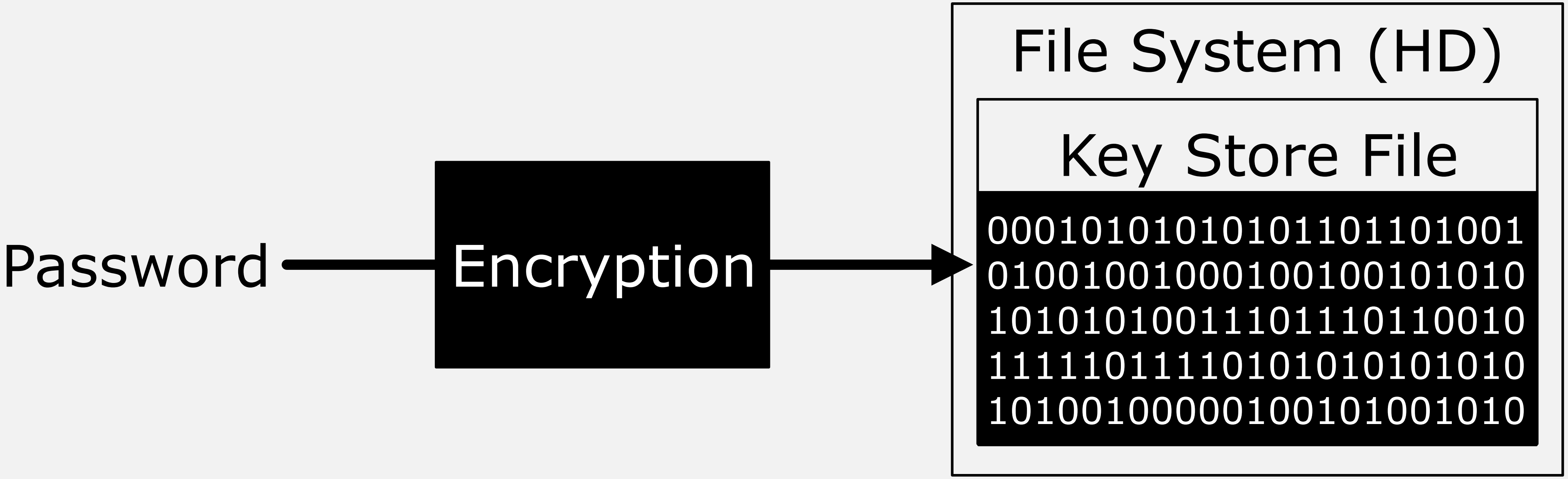
Nail in the Java Key Store coffin

Tobias Ospelt, modzero AG

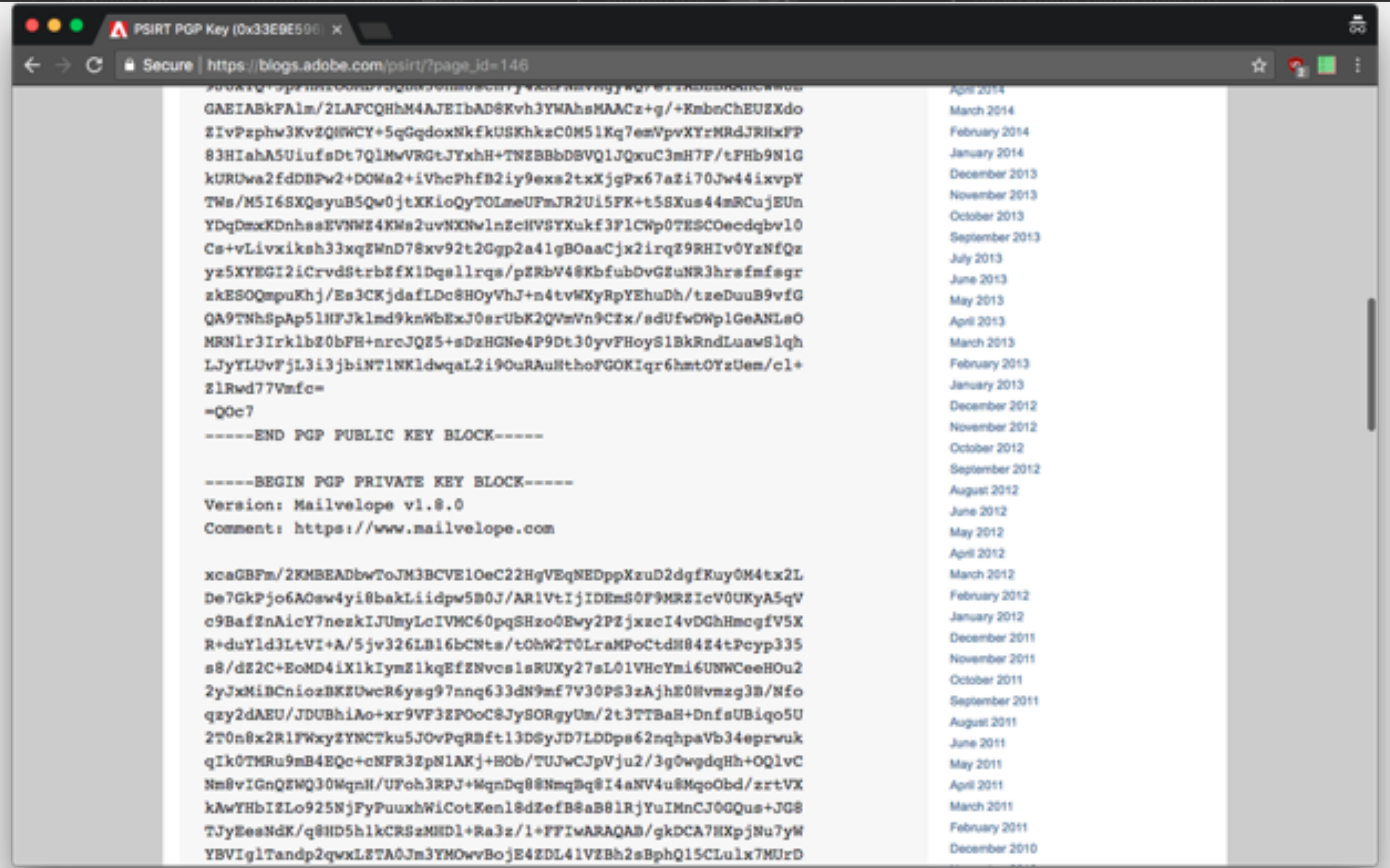
Content

- Purpose/mechanics of Key Store files
- Key Store types
- Purpose/mechanics of JKS
- Weaknesses and Cracking
- Recommendations

Purpose of Key Store files - User view



Not How Asymmetric Crypto Works



Key Store Types

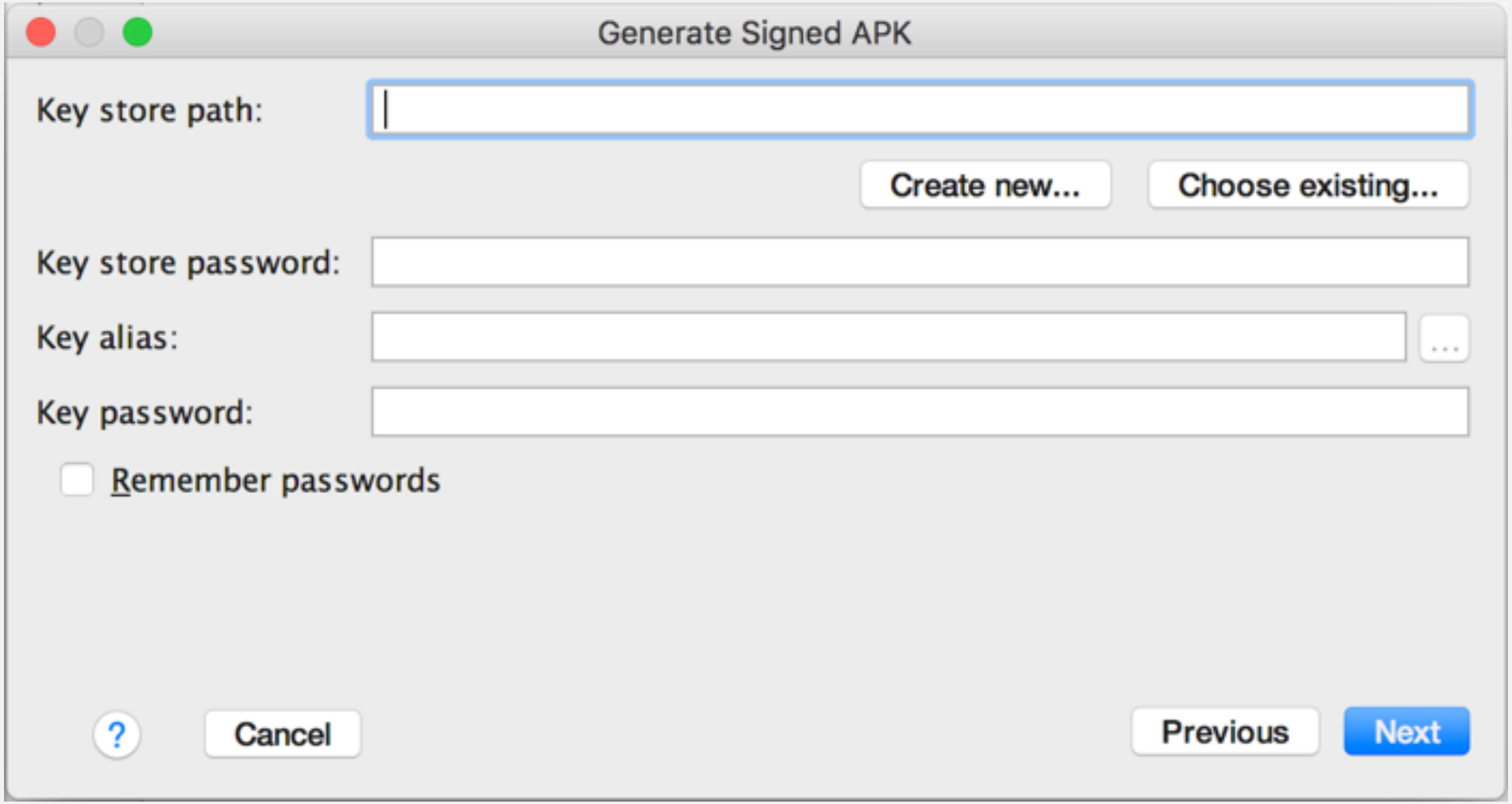
- Various options in Java
  - Java Key Store (JKS)
  - JCEKS
  - BouncyCastle Key Store (BKS)
  - PKCS#12

modzero

Usage of JKS

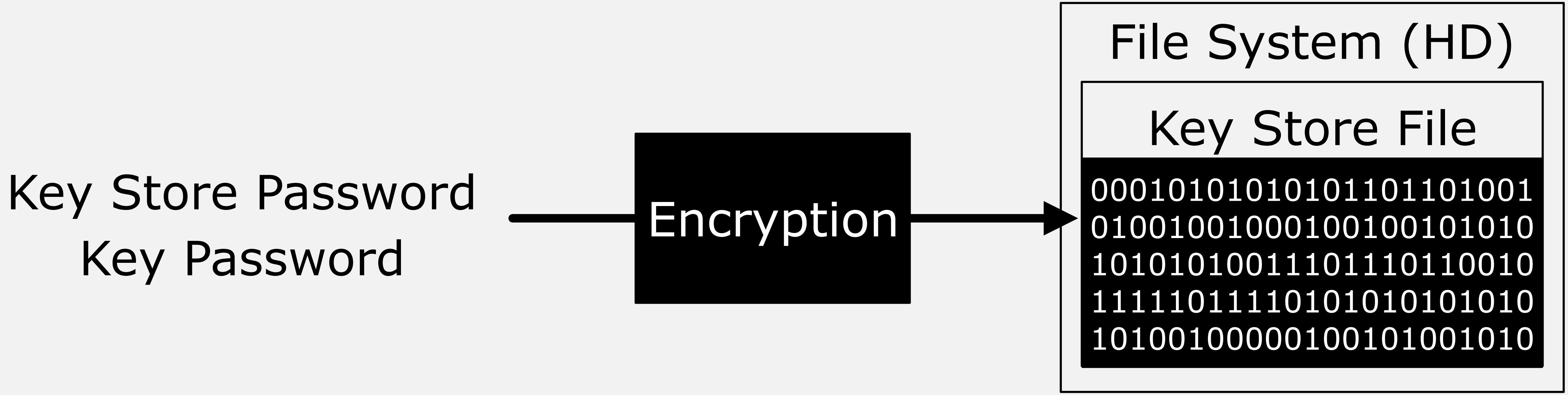
- Default format in all Java and Android versions
- Oracle databases (TLS keys), Apache Tomcat (TLS keys), Android Studio (app signature keys)…
- Java + public key cryptography

Usage of JKS - Android Studio

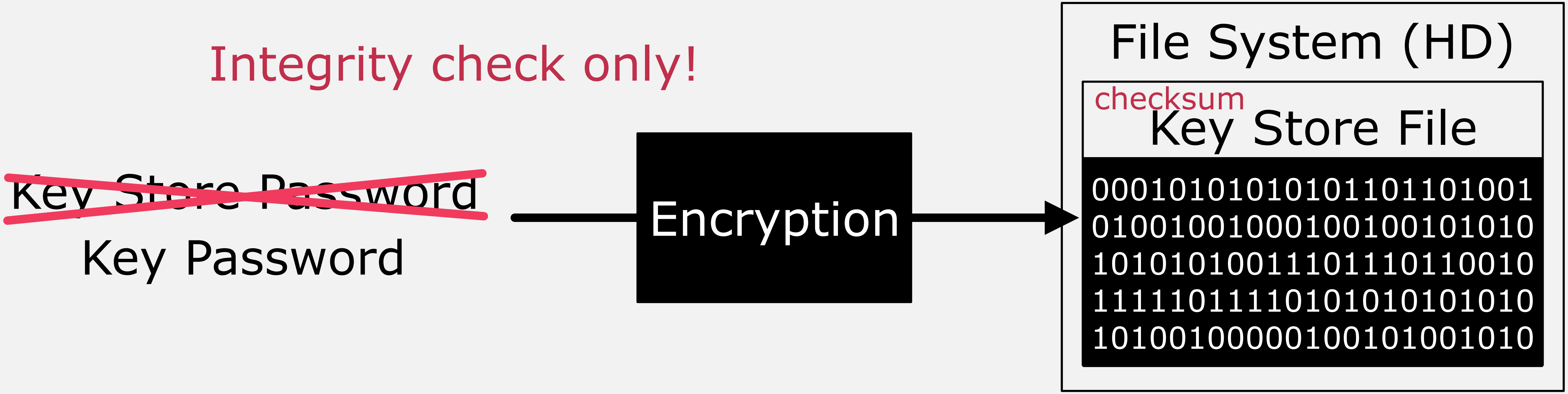


modzero

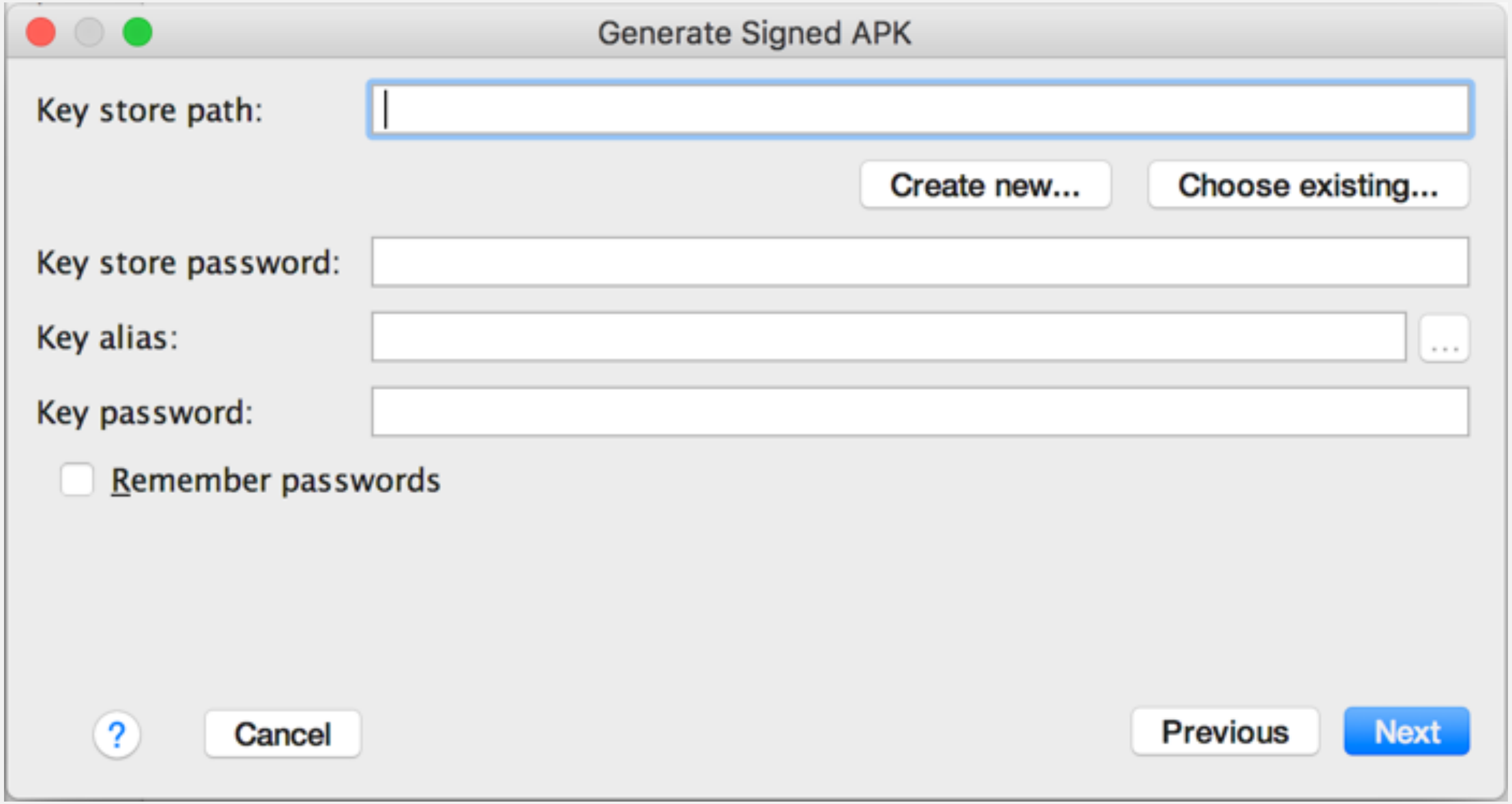
How does JKS Work?



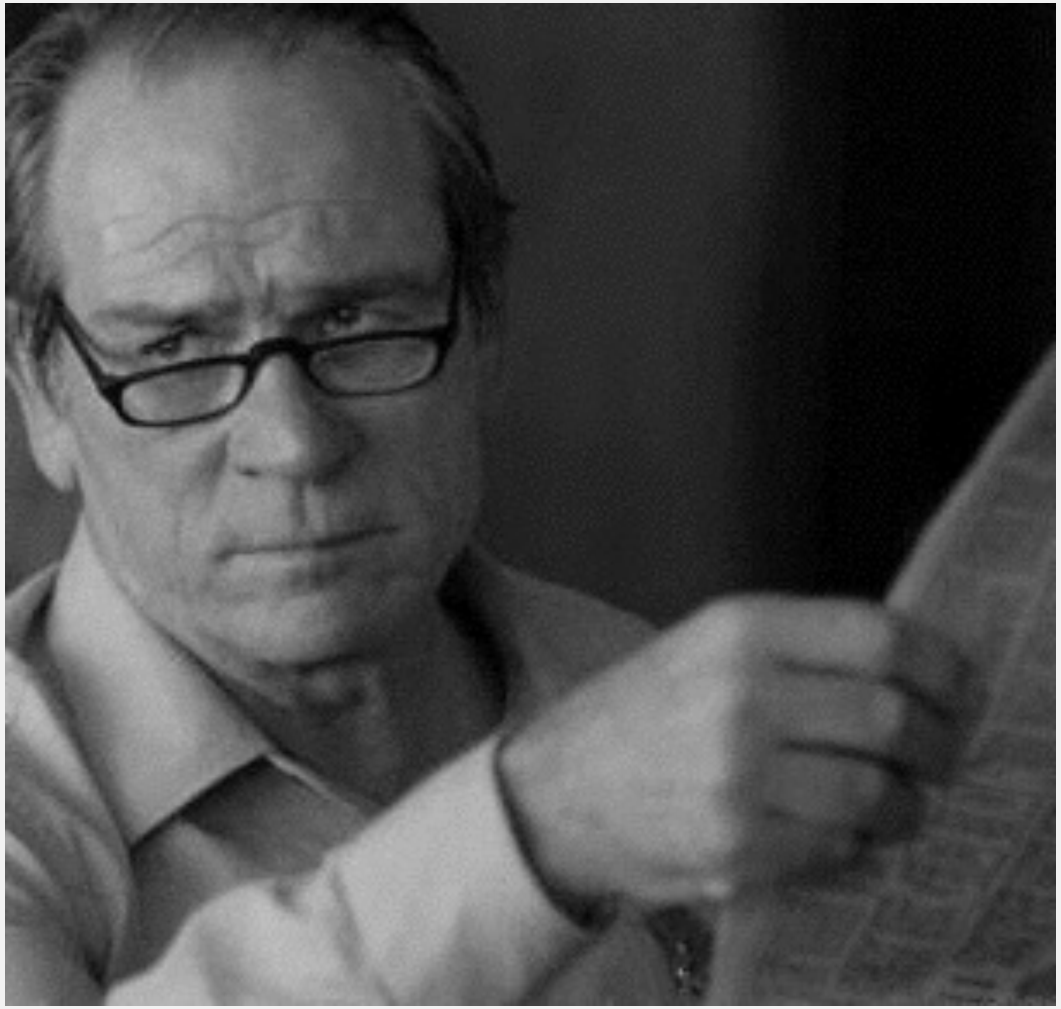
How does JKS Work?



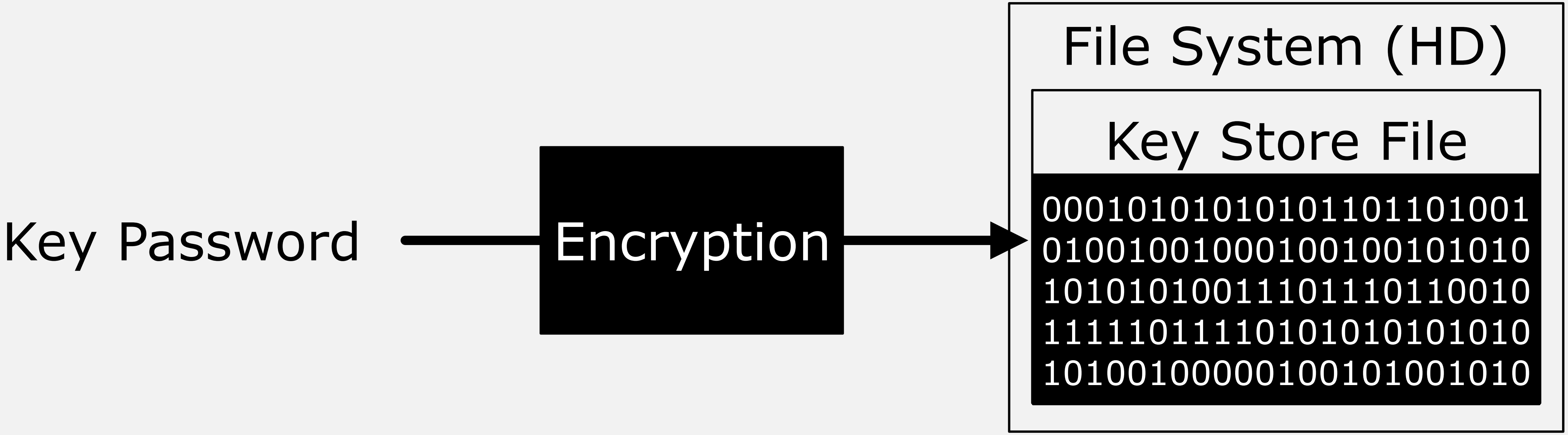
Usage of JKS - Android Studio



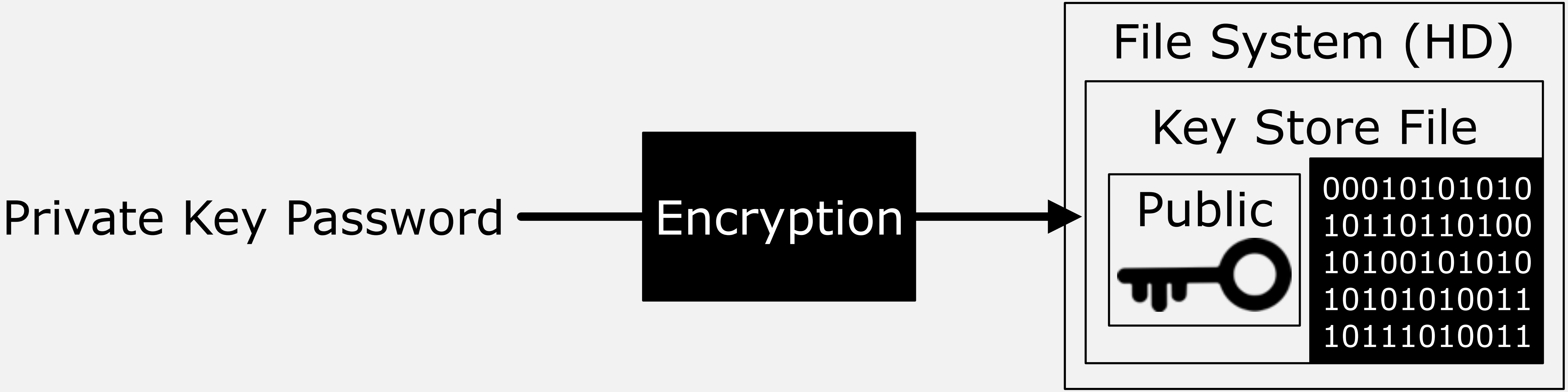
Key Store Password only for Integrity



How does JKS Work?



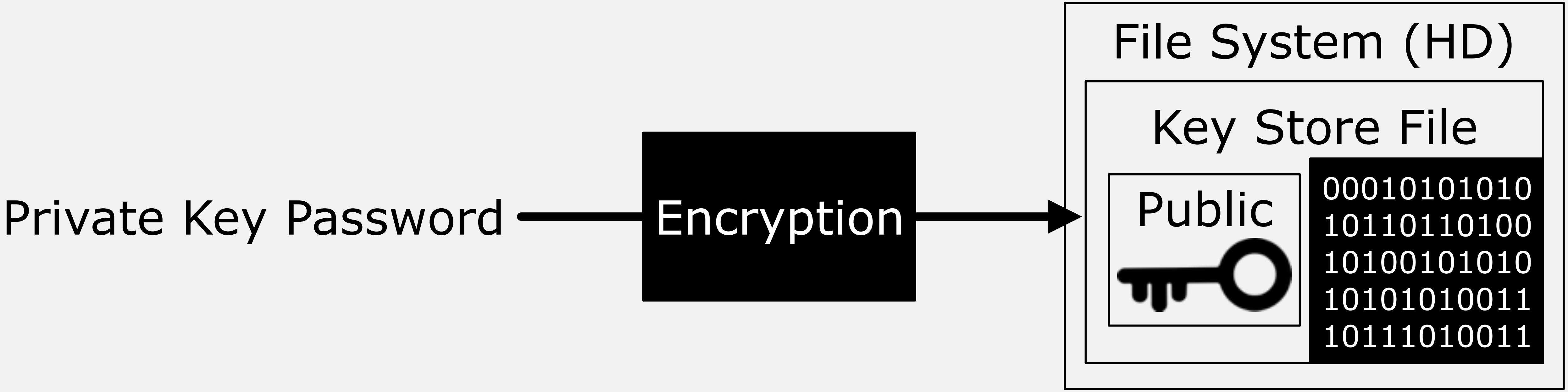
How does JKS Work?



Public Key Not Encrypted

modzero

How does JKS Encryption Work?



modzero

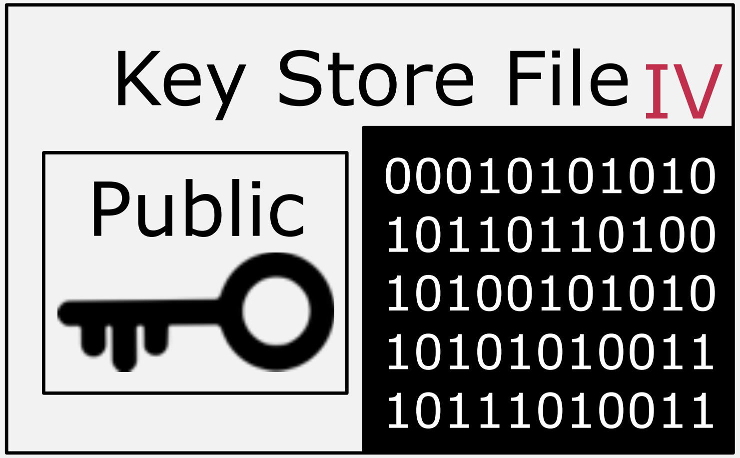
Encryption of the Private Key

- Private Key XOR Key Stream = Encrypted Private Key
- Encrypted Private Key XOR Key Stream = Decrypted Private Key
- How is the Key Stream generated for JKS?

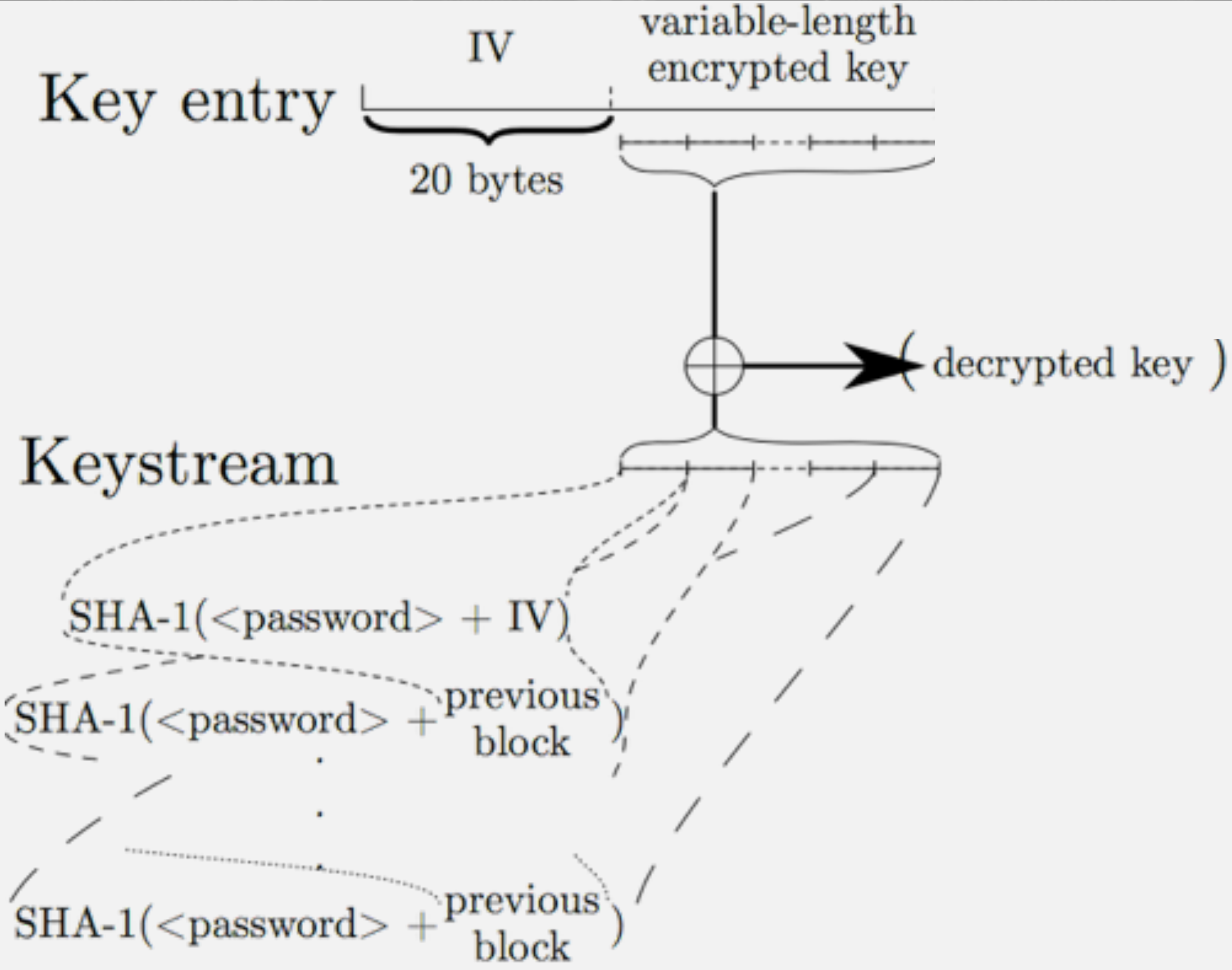
modzero

Key Stream Generation

- Invented a Password Based Encryption (PBE) using SHA-1
- Generating the key stream:
  - A = SHA1(password + IV)
  - B = SHA1(password + A)
  - C = SHA1(password + B)
  - …
  - Key Stream = Concatenate A, B, C…

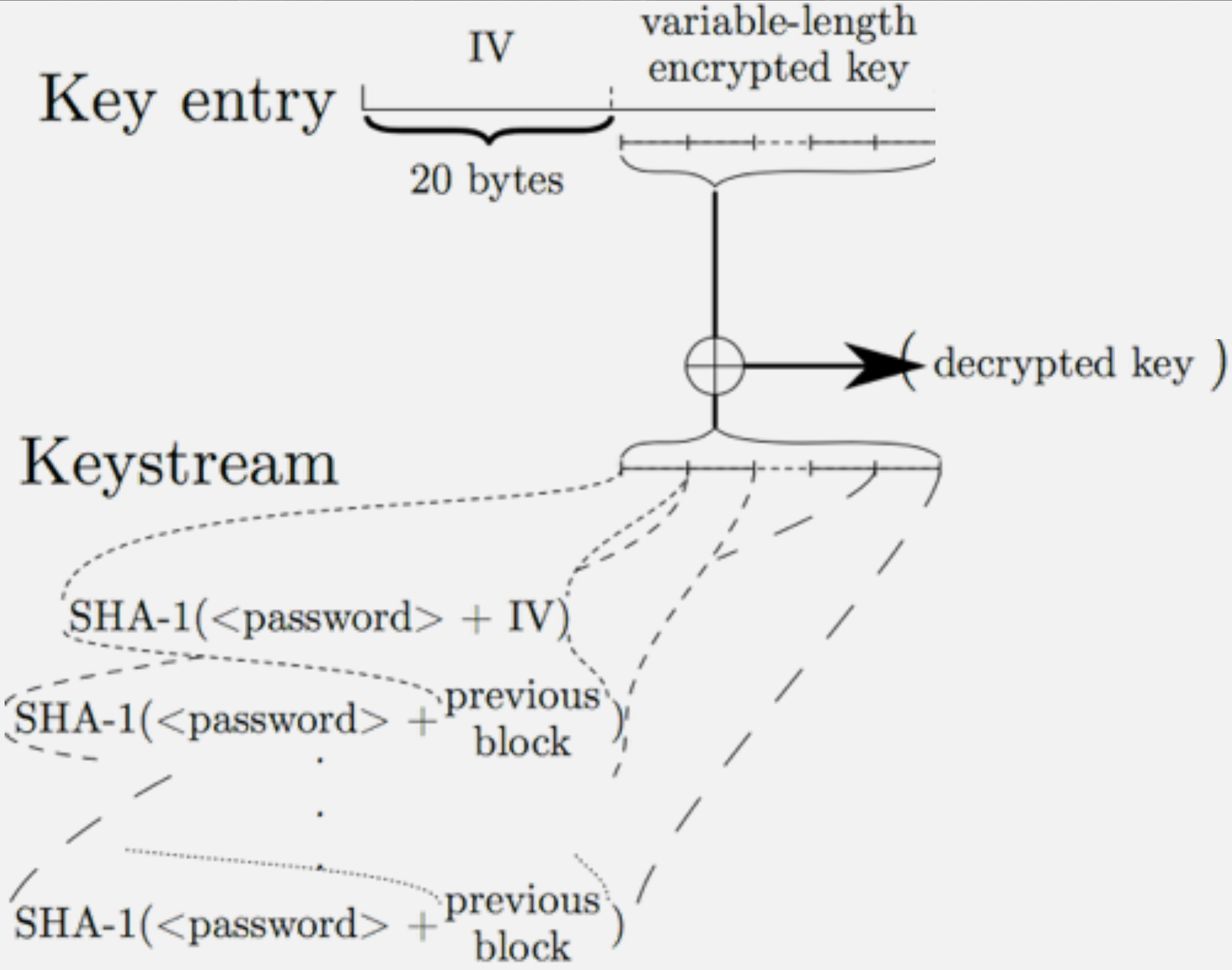


Decryption of the Private Key



modzero

Not so Obvious Weakness



Not so Obvious Weakness

- "*Only one SHA-1 application is required to derive the first keystream byte. Since DER encoded keys contain a lot of structure in their first bytes, […] makes a dictionary-based cracker highly efficient.*" - cryptosense.com
- Cool… but where is the PoC?
  - Is that even feasible in practice?

Not so Obvious Weakness - PoC

- For password cracking, we only need to do:
  1. SHA1(password candidate+IV)
  2. XOR first 20 bytes of encrypted key =
     first 20 bytes of decrypted key
  3. Check first 20 bytes looks like a private key

modzero

The first 20 Bytes of a Decrypted Private Key

- PKCS#8, DER encoding, ASN.1
- In theory:
  - OID 9 bytes long "somewhere at the start"
    - 0x2a864886f70d010101 (rsaEncryption)
    - Best solution: "Search" for OID

The first 20 Bytes of a Decrypted Private Key

- In practice:
  - "Search" for OID is inefficient
  - Let’s look at thousands of private keys and brute force…
    - Lucky: Fixed values 16 out of 20 bytes

```
RSA all: 0x30????????00300d06092a864886f70d010101
DSA 512: 0x30????????3081a806072a8648ce3804013081
DSA rest:0x30????????003082012c06072a8648ce380401
EC (256):0x30????????1306072a8648ce3d020106082a86
…
```

Not so Obvious Weakness - Optimisation

- Example for an RSA key
    1. SHA1(password candidate+IV)
    2. XOR first 20 bytes of encrypted key =
       first 20 bytes of decrypted key
    3. Check if first 20 bytes are

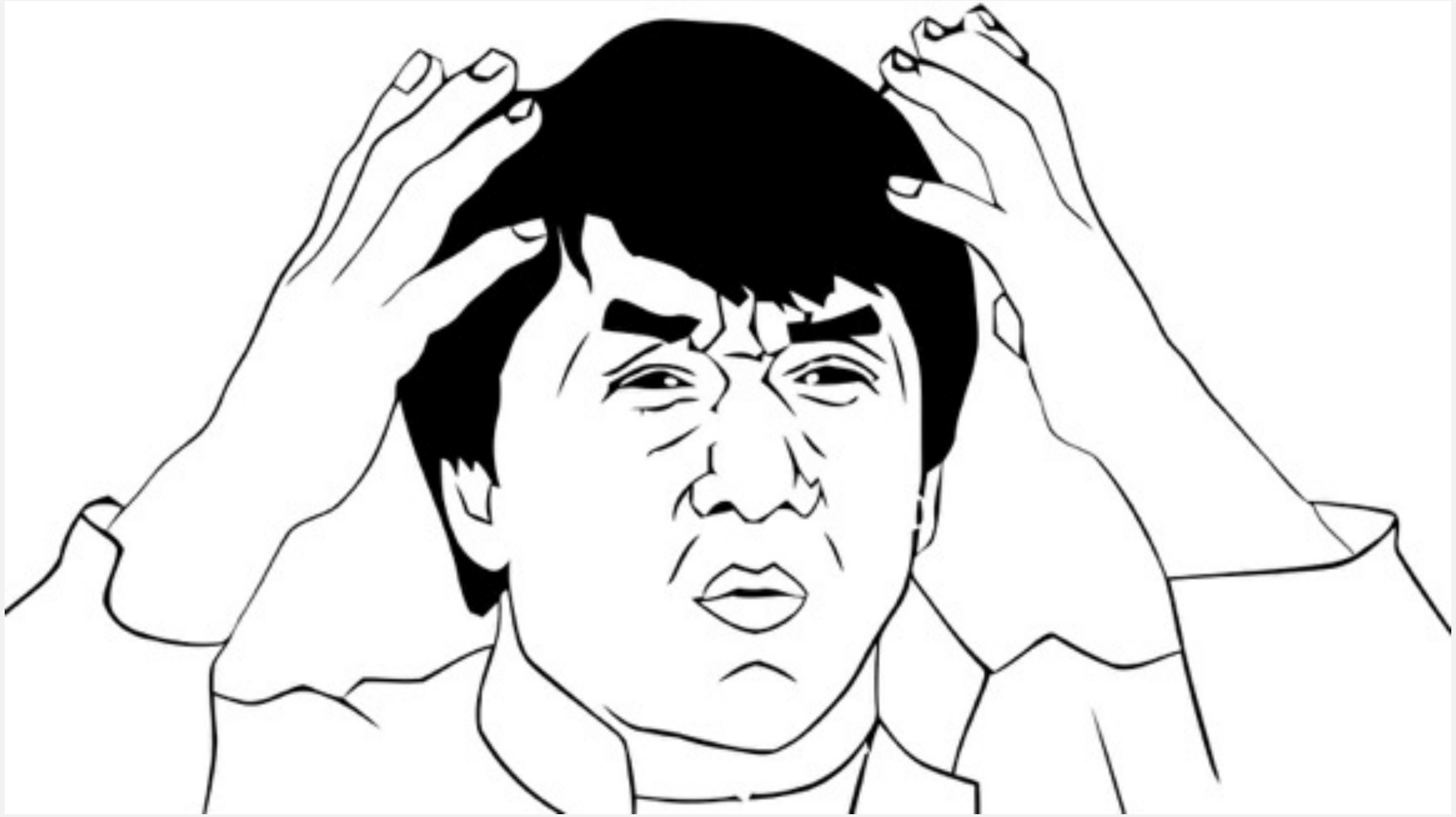
`0x30????????00300d06092a864886f70d010101`

modzero

Not so Obvious Weakness - Result

- Example for an RSA key
  1. SHA1(password candidate+IV)
  2. Check if first 20 bytes correspond to the precalculated 16 bytes
- Implemented in the hashcat password cracker tool to run on GPUs (thanks atom!)
  - It uses a weakness in SHA-1 to be even faster

One SHA-1 calculation for password cracking



modzero

Attacking a JKS File

```
$ java jar JksPrivkPrepare.jar file.jks > hash.txt
$ ./hashcat -m 15500 -a 3 -w 3 hash.txt ?u?u?u?u?u?u?u?u?u
hashcat (v3.6.0) starting...
[…]
* Device #1: GeForce GTX 1080, 2026/8107 MB allocatable, 20MCU
[…]
Hash.Type........: JKS Java Key Store Private Keys (SHA1)
[…]
Speed.Dev.#1.....:  7946.6 MH/s (39.48ms)
[…]
```

Attacking a JKS File

- All alphanumeric passwords of length 8
  - 8 hours on a single NVidia 1080 GPU

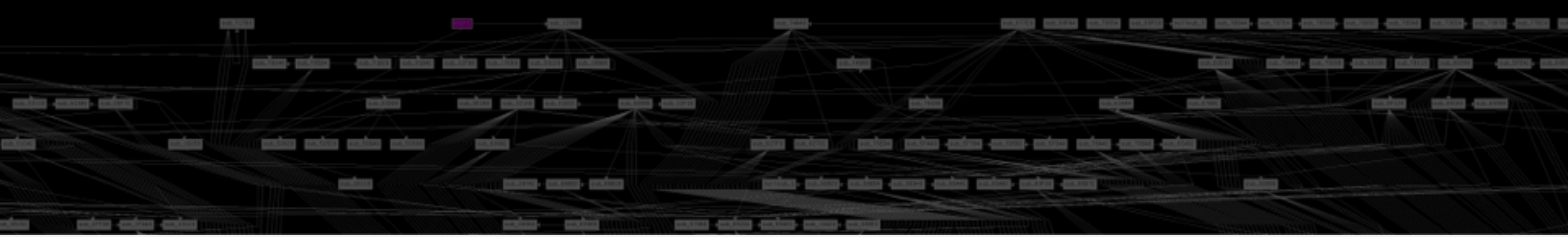
Recommendations

- Never do your own Crypto
- Refactor your Java Software
- Don’t use JKS
  - Good passwords (length 12+), keep file secret
  - PKCS#12 default for upcoming Java 1.9
    - Prediction: JKS will stay for a long time
      - "Existing keystores will not change"
      - "Keystores tend to be long-lived"
- More details in POC||GTFO 0x15 journal

Thank you for your Attention

Questions?

Tobias Ospelt,
tobias@modzero.ch
Twitter: @floyd_ch



Teaser Question 1

- How do you know which fingerprint to expect (RSA, DSA 512, DSA rest, EC, etc.)?

The First 20 Bytes of a Decrypted Private Key

```
RSA all: 0x30????????00300d06092a864886f70d010101
DSA 512: 0x30????????3081a806072a8648ce3804013081
DSA rest:0x30????????003082012c06072a8648ce380401
EC (256):0x30????????1306072a8648ce3d020106082a86
…
```

- But how do you know which fingerprint to expect?
  - Public Key is not encrypted, just check

Teaser Question 2

- You don’t know all twenty bytes of a fingerprint (the question marks), how do you know you didn’t guessed the wrong password?

So many question marks!

`RSA all: 0x30????????00300d06092a864886f70d010101`

- Yes, not 100% probability that the password also matches
- An earlier implementation relied on fewer fixed bytes and had to check if the entire key decrypts properly after finding a candidate…
- But $1/2^{120}$ probability for a failure, which means we never hit it for password brute-forcing

modzero

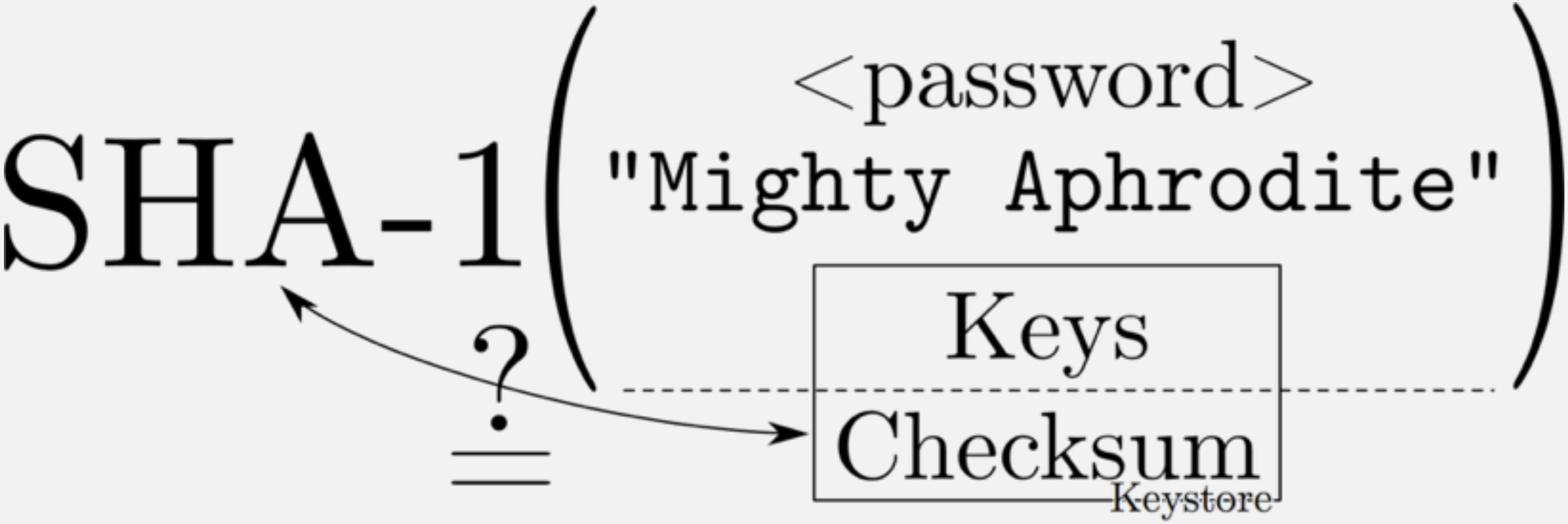
Teaser Question 3

- If no Private Key Password is specified, the Key Store Password is used. Could we attack the Key Store Password then? If no, why not? If yes, why don't we?

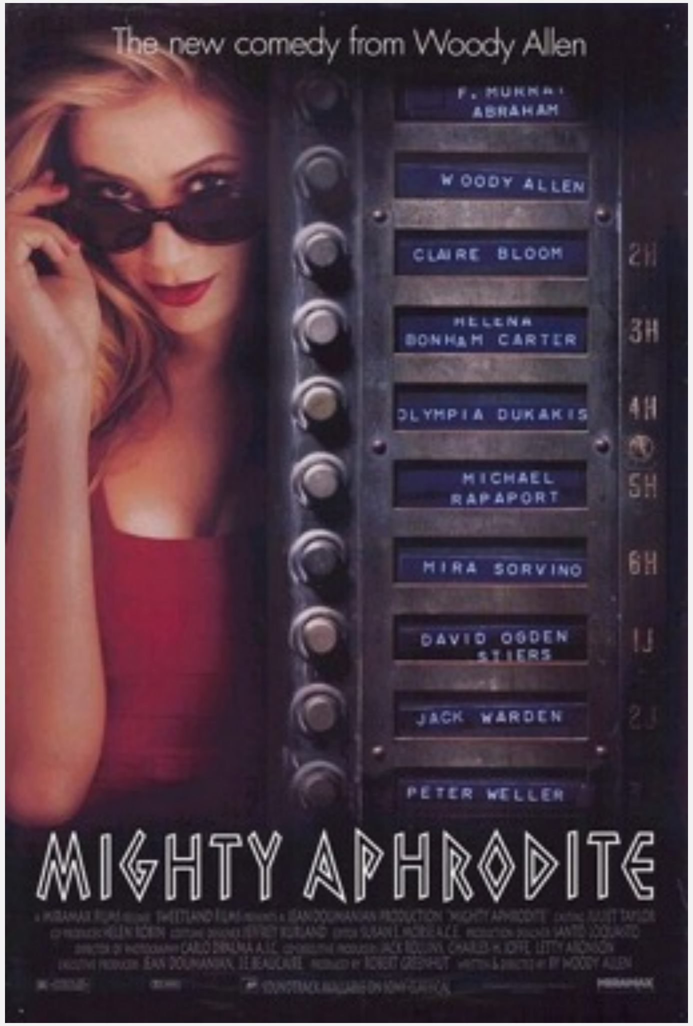
Crack the Key Store Password?

- Default:
  - If no Private Key Password is specified it is set to the same value as the Key Store Password
- If the default case applies (same passwords), we can crack any of them
- Actually nearly all other password crackers do it
  - They crack the wrong password sometimes…

Key Store Password - Integrity Check



Key Store Password - Integrity Check



Why not crack the Key Store Password?

- Which cracking approach has better performance?
- More data go into the SHA-1 calculation, whereas otherwise it is only password+IV
  - Benchmarking showed that cracking the private key directly is more efficient
- Plus it also works in the non-default case (different passwords)