

Nail in the Java Key Store coffin

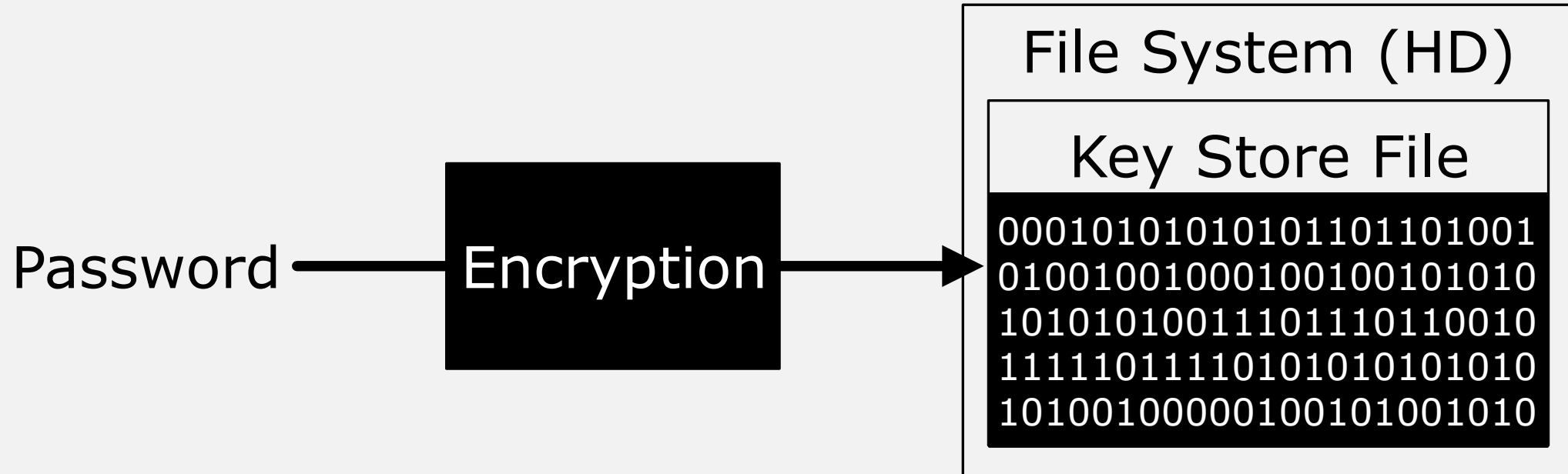
Tobias Ospelt, modzero AG



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

```
PSIRT PGP Key (0x33E9E596) x
Secure | https://blogs.adobe.com/psirt/?page_id=146
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: Mailvelope v1.8.0
Comment: https://www.mailvelope.com

xcaGBFm/2KMBEADbwToJM3BCVE1OeC22HgVEqNEDppXzuD2dgfKuy0M4tx2L
De7GkPjo6AOsw4yi8bakLiidpw5B0J/AR1vtIjIDEmS0F9MRZicV0UKyA5qV
c9BafZnAicY7nezkJJUmYLeIVMC60pqSHzo0Ewy2PZjxzcI4vDghHmcqfV5X
R+duYld3LTVI+A/5jv326LB16bcNts/tOhW2T0LraMPoCtdH8424tPcyp335
s8/dZ2C+eOmd4ix1kIymZ1kqEfZNVcs1sRUXy27sL01VhCYmi6UNWCeeHou2
2yJxmIBcNiozBKZUwCR6ysg97nnq633dN9mf7V30PS3zAjhE0Hvmzq3B/Nfo
qzy2dAEU/JDUBhiAo+xr9VF3EP0oC8Jy8ORgyUm/2t3TTBaH+DnfsUBiqo5U
2T0n8x2R1FWxyZYNCTku5J0vPqRBft13DsyJD7LDDpa62nqhpavb34eprwuk
qIk0TMRu9mB4EQc+cNFR3ZpN1AKj+BOB/TUJwCJpVju2/3g0wgqHh+OQ1vC
Nm8vIGnQEWQ30WqnH/UFoh3RPJ+WqnDq88NmqBq@I4aNV4u@MgoObd/zrtVX
kAwYHbI2Lo925NjPyPuuxhWlCotKen18dZefB8aB81RjYuIhnCJ0QQuS+JG8
TJyEeeNdK/q@HD5h1kCRSzMHD1+Ra3z/1+FFIwARAQAB/gkDCA7HXpjNu7yW
YBVIglTandp2qwxLZTA0Jm3YMOwvBojE4ZDL41VZBh2sBphQ15CLulx7MUrD
-----END PGP PUBLIC KEY BLOCK-----
```

- April 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- October 2013
- September 2013
- July 2013
- June 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012
- August 2012
- June 2012
- May 2012
- April 2012
- March 2012
- February 2012
- January 2012
- December 2011
- November 2011
- October 2011
- September 2011
- August 2011
- June 2011
- May 2011
- April 2011
- March 2011
- February 2011
- December 2010



Key Store Types

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

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

Generate Signed APK

Key store path:

Key store password:

Key alias:

Key password:

Remember passwords

How does JKS Work?

Key Store Password
Key Password

Encryption

File System (HD)

Key Store File

```
00010101010101101101001
01001001000100100101010
10101010011101110110010
11111011110101010101010
10100100000100101001010
```


How does JKS Work?

Integrity check only!

~~Key Store Password~~
Key Password

Encryption

File System (HD)

checksum

Key Store File

```
00010101010101101101001
01001001000100100101010
10101010011101110110010
11111011110101010101010
10100100000100101001010
```

Usage of JKS - Android Studio

Generate Signed APK

Key store path:

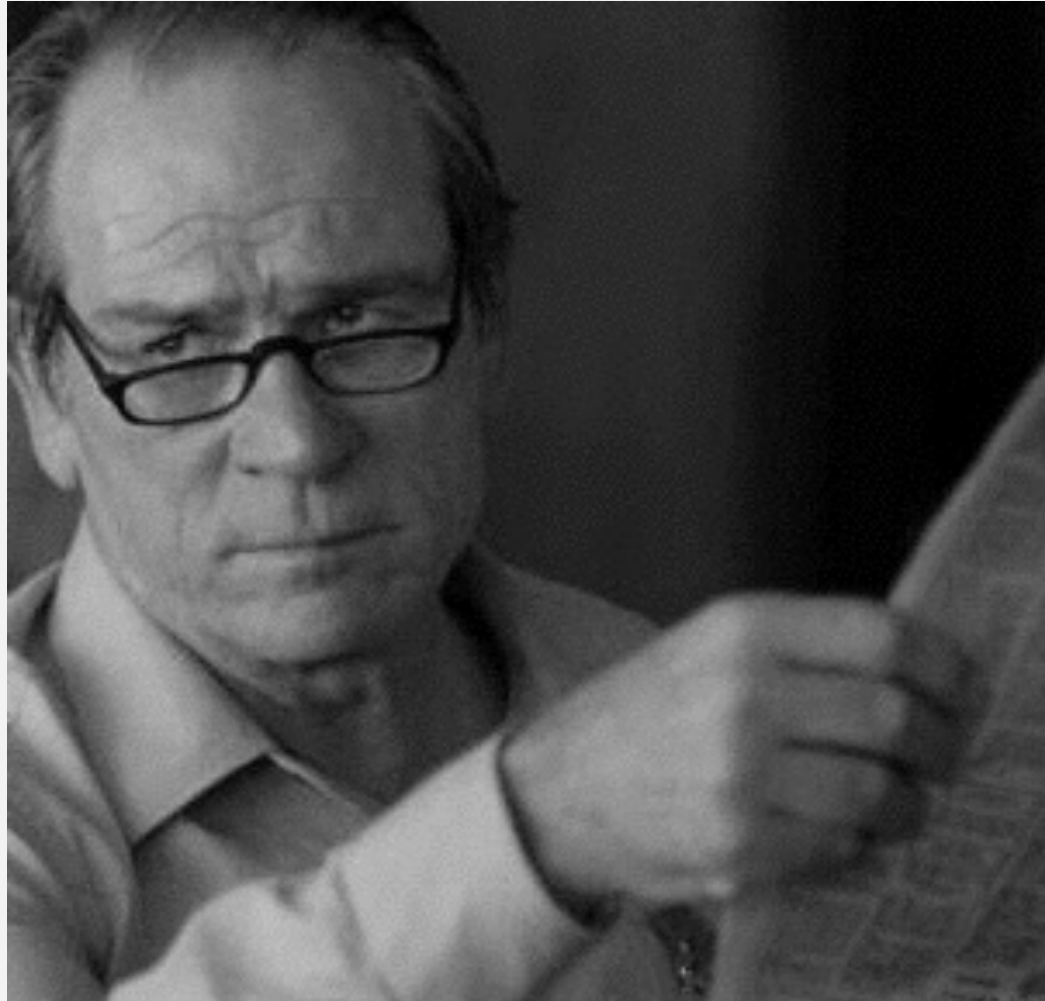
Key store password:

Key alias:

Key password:

Remember passwords

Key Store Password only for Integrity



How does JKS Work?

Key Password

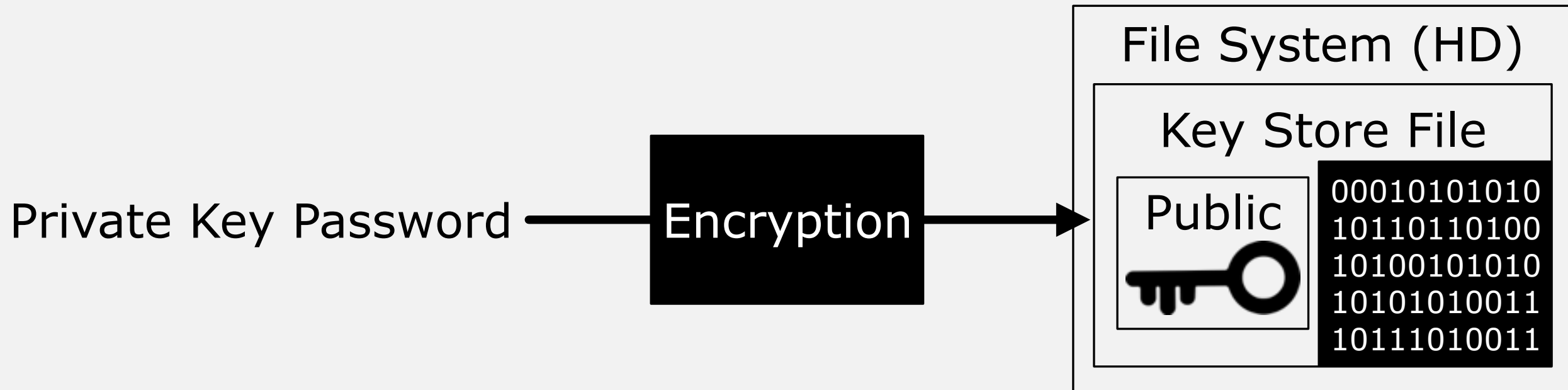
Encryption

File System (HD)

Key Store File

```
00010101010101101101001
01001001000100100101010
10101010011101110110010
11111011110101010101010
10100100000100101001010
```

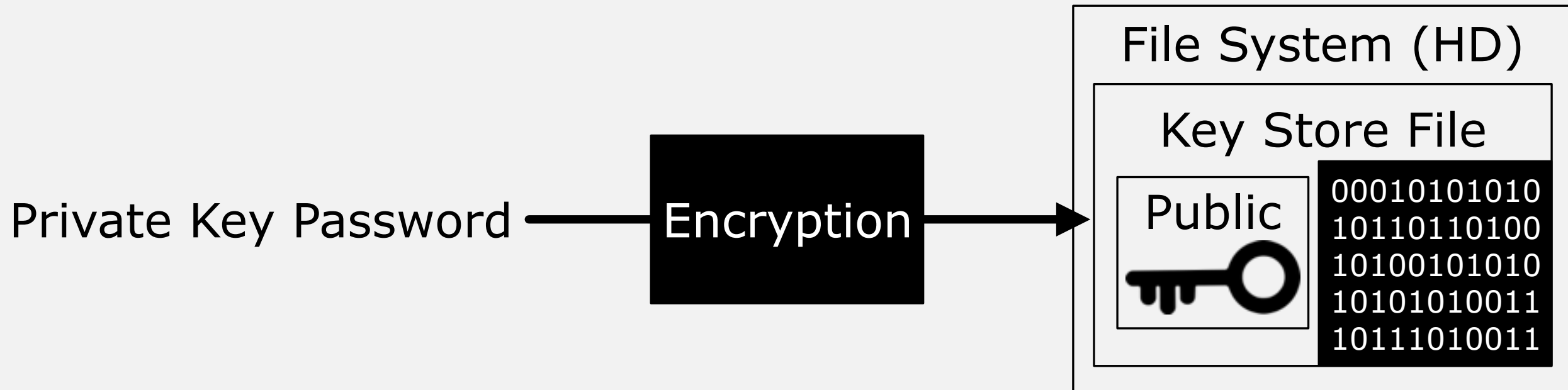
How does JKS Work?



Public Key Not Encrypted



How does JKS Encryption Work?



Encryption of the Private Key

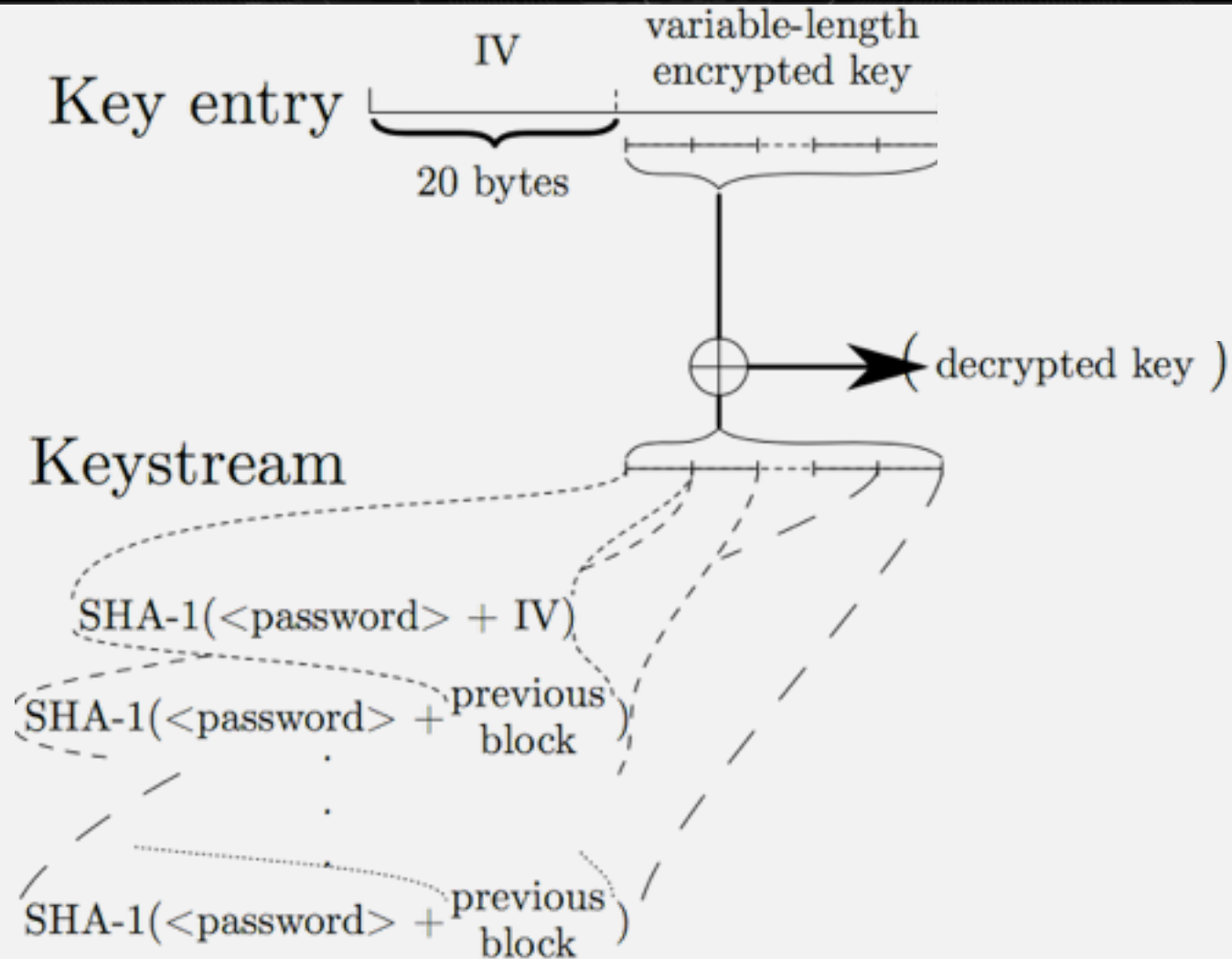
- Private Key XOR Key Stream = Encrypted Private Key
- Encrypted Private Key XOR Key Stream = Decrypted Private Key
- How is the Key Stream generated for JKS?

Key Stream Generation

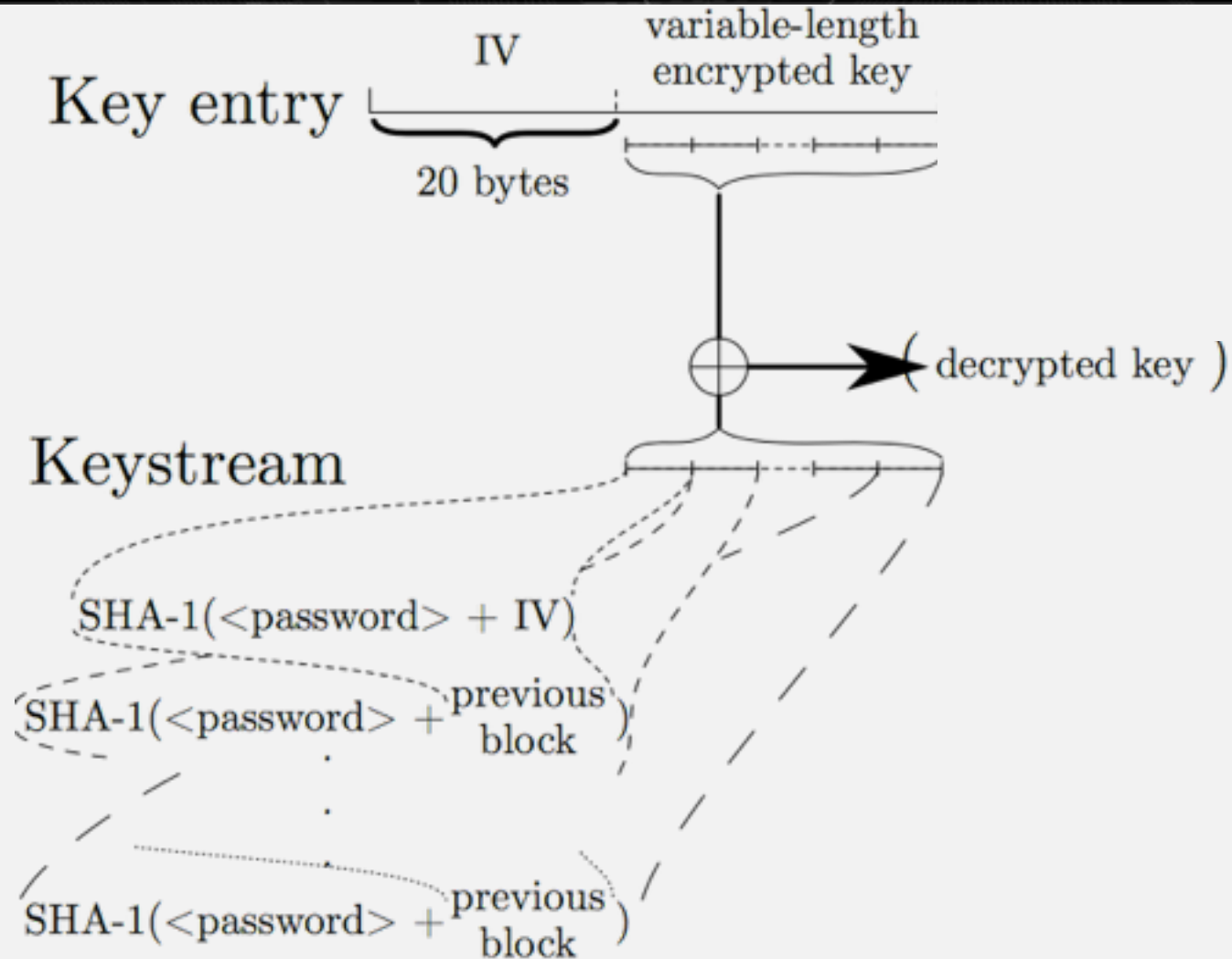
- Invented a Password Based Encryption (PBE) using SHA-1
- Generating the key stream:
 - $A = \text{SHA1}(\text{password} + \text{IV})$
 - $B = \text{SHA1}(\text{password} + A)$
 - $C = \text{SHA1}(\text{password} + B)$
 - ...
 - Key Stream = Concatenate A, B, C...



Decryption of the Private Key



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

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

...

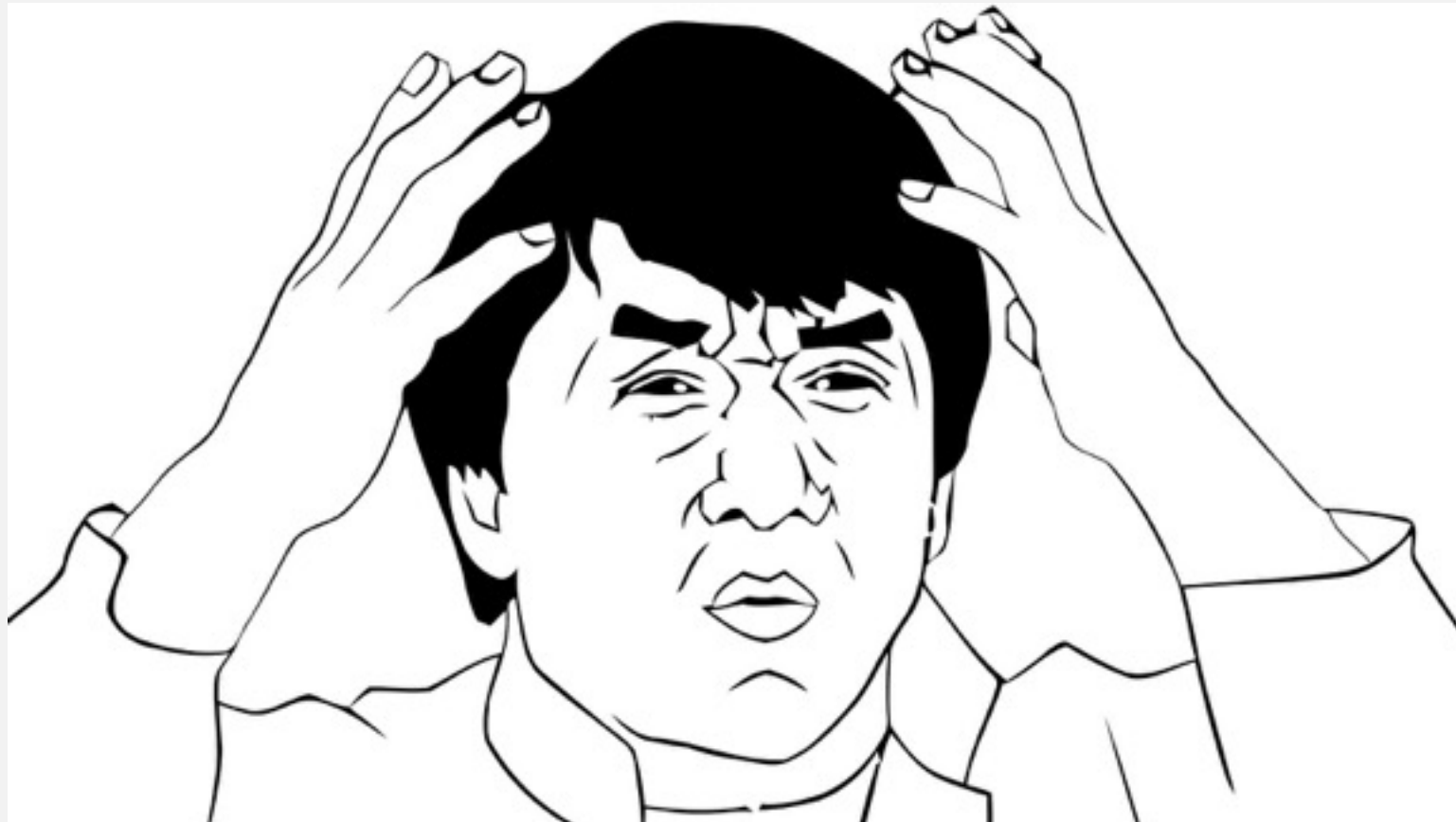
Not so Obvious Weakness - Optimisation

- 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`

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



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
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

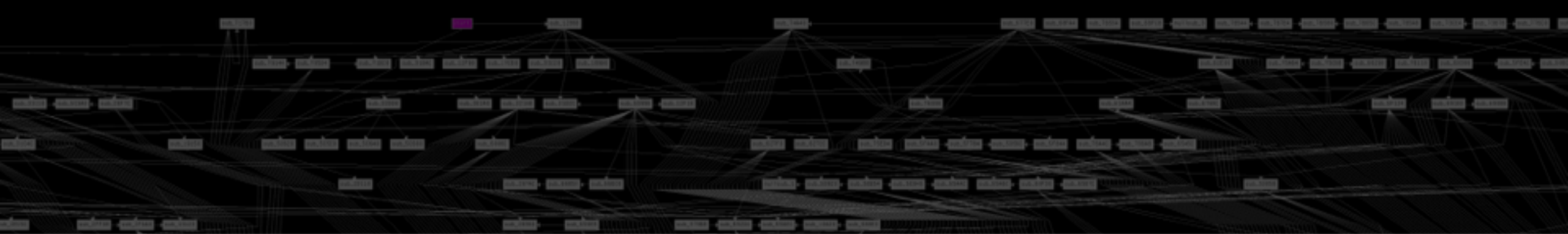
- Never do your own Crypto
- Refactor your Java Software
- Don't use JKS
 - Good passwords (length 12+), keep file secret
 - PKCS#12 default for upcoming Java 1.9
 - Prediction: JKS will stay for a long time
 - "Existing keystores will not change"
 - "Keystores tend to be long-lived"
- More details in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

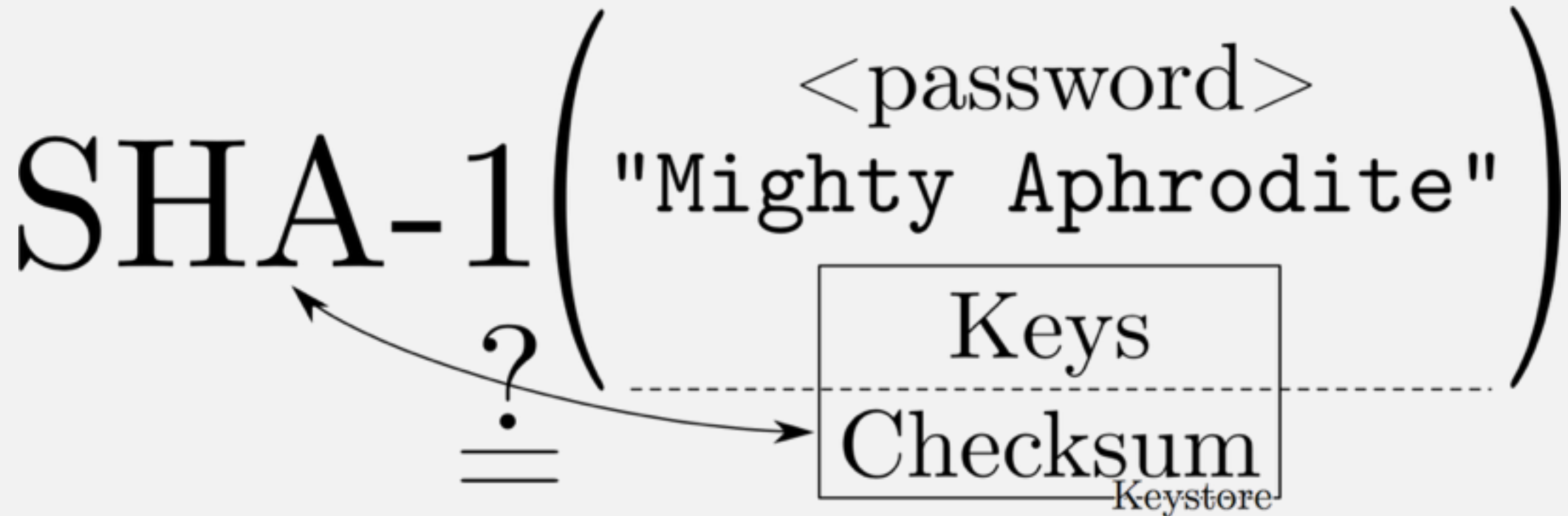
Teaser Question 3

- 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?

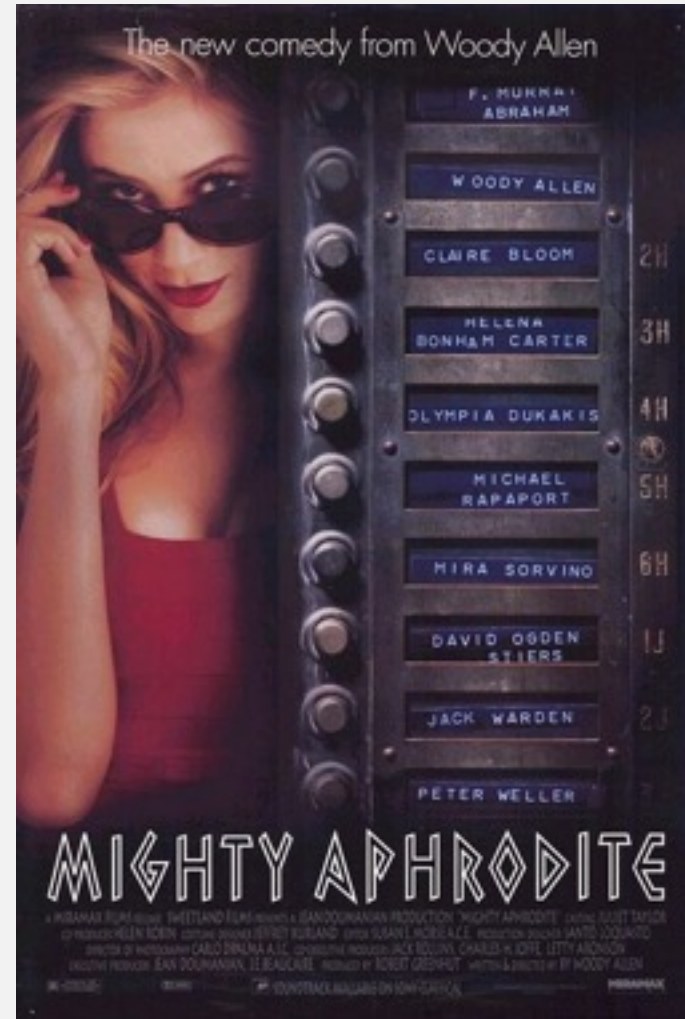
Crack the Key Store Password?

- 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)