Key Extraction Attacks through Encrypted Private Key Corruption

Lara Bruseghini, Kenny Paterson, Daniel Huigens
Key corruption in insecure storage

Threat model:

- Attacker has write access to encrypted private long-term key of the victim
- The victim uses the key as long as it decrypts successfully

Insecure storage examples: key in transit, stored on the cloud, on USB drive..
Key corruption in insecure storage

Attacker can corrupt encrypted private long-term key of the victim

Klíma and Rosa (2001) show how to target DSA keys:

1) attacker overwrites some public params
2) waits for the victim to sign using the corrupted key
3) extracts secret values from the faulty signature
Key corruption in insecure storage

Attacker can corrupt encrypted private long-term key of the victim

Our findings — any key is potentially vulnerable to faulty signature attacks (& more):

- direct attacks against DSA, EdDSA, RSA keys
- indirect attacks against ECDSA/ECDH/EdDSA and ElGamal keys (can be converted into DSA ones)
Existing protocol-level protections

- Private values encrypted either with AEAD or CFB (always authenticated)
- No integrity protection over public fields → decryption won’t fail
- Key binding signatures can be forged/replaced by the attacker
- However:
  - checking fingerprint reveals public key corruption
  - third-party certifications won’t be verifiable

Our threat model: user/app might trust a key as long as it decrypts
Key validation in implementations

We have found that many libraries implement some key checks, but not always effective

- Trial signature or decryption
- Algorithm-specific checks

\[ n = pq \]

\[ de \equiv 1 \mod (p - 1)(q - 1) \]
Key validation issues

ElGamal: full validation is infeasible

DSA: difficult and expensive to validate

EdDSA: expensive to validate

In practice:

- none of the libraries we have reviewed is fully safe against attacks
- we found two real-world apps where this vulnerability was exploitable
Possible protocol-level solution

Our proposal:

- for AEAD-encrypted keys, put public key in the Associated Data
- for CFB-encrypted keys, hash the public key together with private one

Advantages over implementation-level key validation:

- works for any key algorithm
- much faster
- avoids all key validation pitfalls