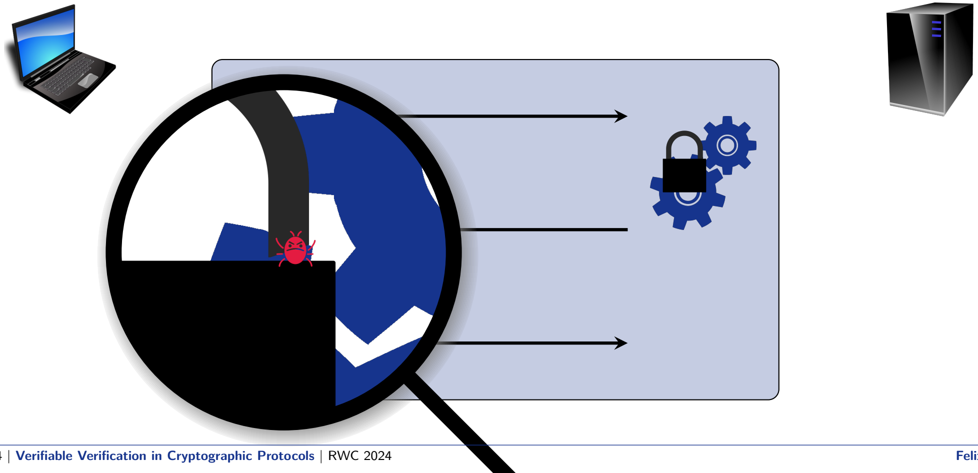
Verifiable Verification in Cryptographic Protocols

Felix Günther
IBM Research Europe – Zurich

joint work with **Marc Fischlin** (TU Darmstadt)

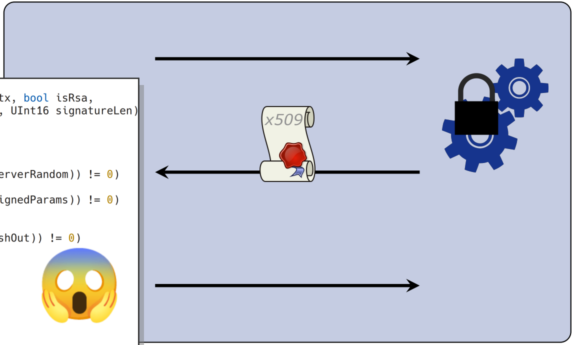
When Locks Fail. . .

When Crypto Locks Fail. . .



When Crypto Locks Fail. . .

. . . in Practice




```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
    SSLBuffer signedParams, uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

Apple `goto fail;`

(Why) Does Cryptography Have to Be So Brittle?

- **Verification**

validating signatures	✗	Apple `goto fail;`, GnuTLS, curl
validating MACs	?	OpenSSH generic-EtM
validating curve parameters	✗	small subgroup attacks, Bluetooth fixed coordinate

- **Randomness**

bad RNGs	✗	Debian OpenSSL, Android SecureRandom
bad randomness	✗	Sony Playstation 3, bitcore

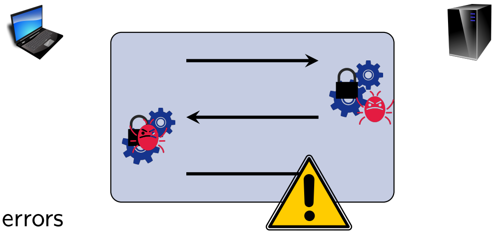
- **Encryption**

when talking to others	✓	
when talking to yourself	✗	AWS zero-key encryption of TLS session tickets

Tying Security to Functionality

Our goal: tie **security** to **basic functionality**

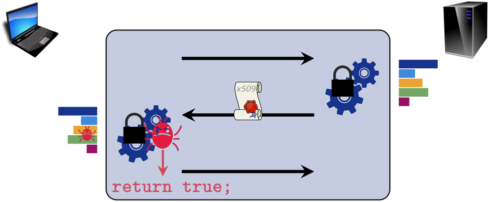
[Heninger @ WAC2, 2019]



- What if...
  - ... we can make **crypto bugs**
  - ... surface through **functional errors**?
- We want to catch **accidental** implementation errors
  - ... by making them **detectable in interop tests**
  - (we cannot prevent malicious implementations — and don't intend to)

Introducing Confirmation Codes

- What if instead of a decision bit, we'd output a **description of essential steps** carried out?



- verification steps: compute & compare intermediate values
- collect relevant intermediate values in a "**confirmation code**"

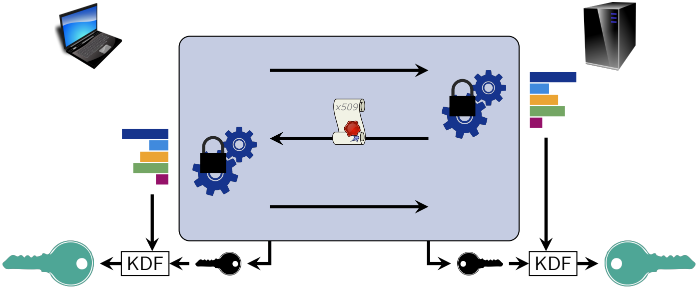
- bugs (like skipping, misinterpreting, input error) → change in confirmation code

- choose confirmation codes carefully:
  - meaningful: careful notion of **unpredictability** (details see paper)
  - low overhead
  - sender (e.g., signer) also able to compute them

Making Cryptographic Protocols Fail Noticeably

- We get: sender + receiver agree on confirmation code $\Longrightarrow$ verification followed necessary steps
- ... so let's have both check they agree? $\rightarrow$ yet another verification step...

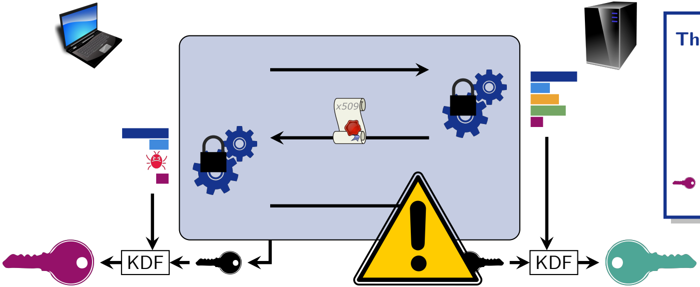
- Better: use confirmation codes in overall protocol — here: **secure connection establishment**



Making Cryptographic Protocols Fail Noticeably

- We get: sender + receiver agree on confirmation code $\Longrightarrow$ verification followed necessary steps
- … so let's have both check they agree? $\rightarrow$ yet another verification step…

- Better: use confirmation codes in overall protocol — here: **secure connection establishment**



Theorem: (details see paper)

unpredictable

+

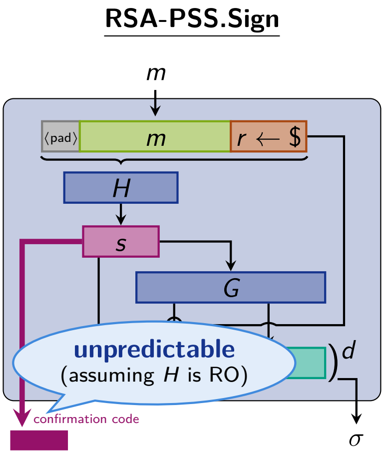
KDF collision-resistant

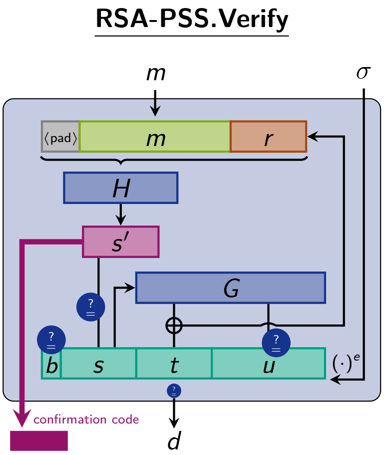
$\Downarrow$

$\neq$ **noticeably incorrect** under **faulty verification**

Adding Confirmation Codes to Crypto Schemes

Example: RSA-PSS Signatures [PKCS #1 v2.1, NIST FIPS 186-5]





Adding Confirmation Codes to Crypto Schemes

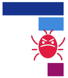
- **Verification**, made **verifiable**

validating signatures	✓	RSA-PSS
validating MACs	✓	HMAC
validating curve parameters	✓	validity and subgroup checks for elliptic curve points

- for each, **we prove**
  - confirmation code **unpredictability**
    the ingredient to make them noticed in protocols (e.g., failing connections)
  - confirmation codes don't hurt **regular security**
    easy for asymmetric and public verification, but secret-keyed primitives (HMAC) require care

Deployment Discussion

- **Deployment** of confirmation codes requires system-level efforts & design discussions
  - **API**: might surface confirmation codes optionally, for backwards compatibility
  - **Live vs static**: confirmation codes for online signatures are produced live by the signer – how to best integrate confirmation codes of static signatures (e.g., in certificates)?
  - **Transient**: should one be able to (de)activate the use of confirmation codes? (think: TLS extension)
- Confirmation codes need to be **used with care**
  - leaking details about errors within an implementation might leak information about inputs
  - safest to be consumed by a higher-level protocol (e.g., KDF), not exposed



- Ultimately, confirmation codes are meant to detect flaws prior to **deployment in production**

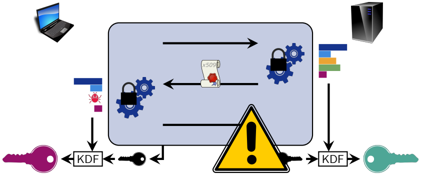
Further Directions

- Many **more candidates** for verifiable verification
  - **primitives**:
    - authenticated encryption
    - FO-based KEMs
    - verifiable secret sharing
    - ...
  - **protocols**:
    - code signing
    - secure messaging
    - entity authentication
    - ...
- ... and the idea of **tying security to basic functionality** is not restricted to verification

Summary

We

- introduce **confirmation codes** for verification
  to **tie security to basic functionality**
- present intuitive (and provably secure) confirmation codes
  for **RSA-PSS, HMAC, curve point validation**
- exemplify their usage in key exchange protocols
  to make **secure connections fail noticeably**
- think the basic idea is **applicable more broadly**,
  and are happy to **discuss deployment**



full version @ IACR ePrint: `https://ia.cr/2023/1214`

Thank You!

mail@**felixguenther.info**