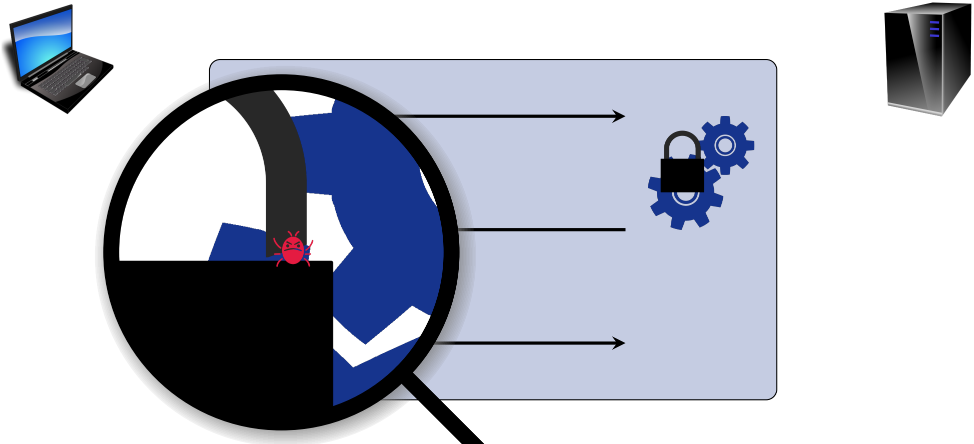
Verifiable Verification in Cryptographic Protocols

Felix Günther
IBM Research Europe – Zurich

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

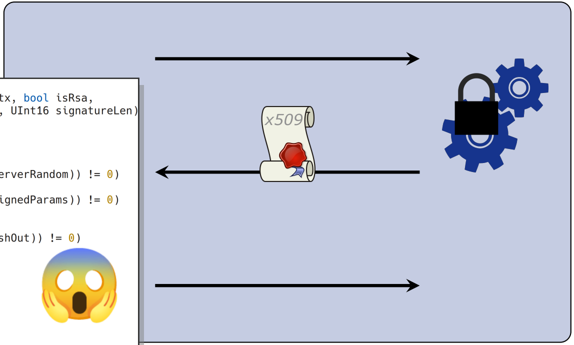
When Locks Fail. . .

When Crypto Locks Fail. . .



When Crypto Locks Fail. . .

. . . in Practice




```c
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

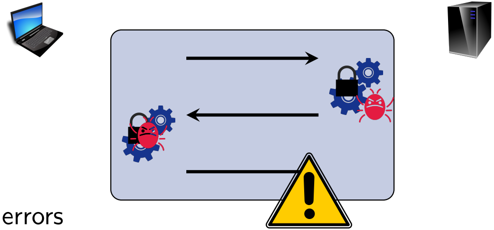
- **Encryption**

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)

In This Talk

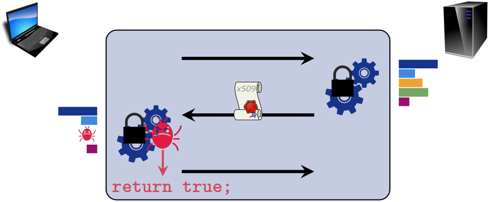
Our goal: tie **security** to **basic functionality**

[Heninger @ WAC2, 2019]

- introduce **confirmation codes** to surface bugs in verification implementations
- present intuitive (and provably secure) confirmation codes
  for **RSA-PSS, HMAC, curve point validation**
- tie them to basic functionality in **secure connections**
- discuss **further directions** & system-level efforts for **deploying confirmation codes** in practice

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**
  - low overhead
  - sender (e.g., signer) also able to compute them

Introducing Confirmation Codes

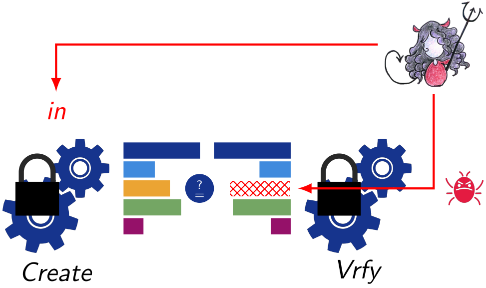
Defining Verifiable Verification

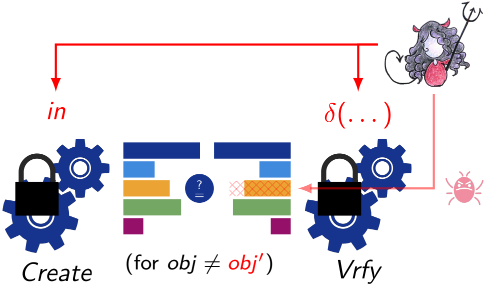
	KGen			Creation	Verification
Signatures (asymmetric)	sk		pk	$Sign(sk, m) \to \sigma$	$Vrfy(pk, m, \sigma) \to d$
MACs (symmetric)	sk	vk		$Tag(sk, m) \to \tau$	$Vrfy(vk, m, \tau) \to d$
EC point validity (public)			pp	sample point (x, y)	$d := [x^3 + ax + b = y^2]$
Verification	ck	vk	pk	$Create(ck, pk, in) \to (obj, tok)$	$Vrfy(vk, pk, obj, tok) \to d$
Verification w/ confirmation	ck	vk	pk	$Create(ck, pk, in) \to (obj, tok, \vec{c})$	$Vrfy(vk, pk, obj, tok) \to (d, \vec{c})$

Introducing Confirmation Codes

Confirmation-Code Unpredictability

- we want: **unpredictable** confirmation codes wrt. accidental verification errors
- impossible against a malicious implementation of Vrfy → can just compute the right code





(drawings by *Giorgia Azzurra Marson*)

- capture **programming errors**
  (e.g., `goto fail;` skipping verification steps)
- capture **input errors** (+ programming errors)
  (e.g., reading y-coordinate as 0 + not validating)

Adding Confirmation Codes to Crypto Schemes

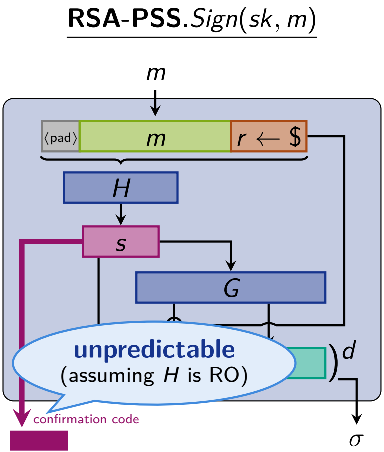
- **Verification**, made **verifiable**

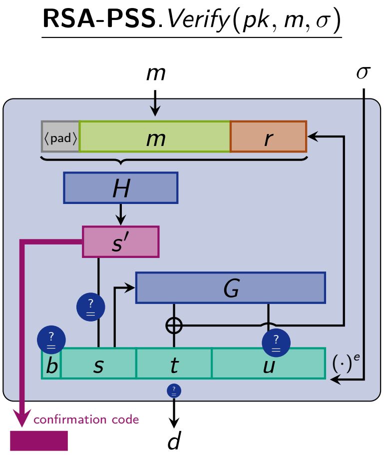
validating signatures	✓	RSA-PSS
validating MACs	✓	HMAC
validating curve parameters	✓	validity and subgroup checks for elliptic curve points

- 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

Adding Confirmation Codes to Crypto Schemes

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





Adding Confirmation Codes to Crypto Schemes

Example: HMAC Message Authentication Code [RFC 2104, NIST FIPS 198-1]

<u>**HMAC**.*Tag*(ck, m)</u>	<u>**HMAC**.*Verify*(vk, m, τ')</u>
$\tau := H(ck \oplus \texttt{opad}, \underbrace{\boxed{H(ck \oplus \texttt{ipad}, m)}}_{\text{confirmation code}})$	$\tau' \stackrel{?}{=} H(vk \oplus \texttt{opad}, \underbrace{\boxed{H(vk \oplus \texttt{ipad}, m)}}_{\text{confirmation code}})$

- **unpredictable**: assuming compression function h is a dual-PRF (as for basic HMAC security)
- **still secure**: [GPR14] PRF proof for HMAC allows that adversary learns inner hash evaluations
- implicit verification: one can also **not send** τ and instead use it as confirmation code

[GPR14] Peter Gaži, Krzysztof Pietrzak, and Michal Rybár. The exact PRF-security of NMAC and HMAC. CRYPTO 2014.

Adding Confirmation Codes to Crypto Schemes

Example: Checking Validity of a Random Point on an Elliptic Curve

<u>KGenC():</u>

pick curve $(a, b, P, q, \mathbb{F})$
$ck \leftarrow vk \leftarrow \perp$
$pk \leftarrow (a, b, P, q, \mathbb{F})$
return (ck, vk, pk)

<u>CreateC(ck, pk, in):</u>

pick $r \xleftarrow{\$} \mathbb{Z}_q$
compute $(x_r, y_r) \leftarrow rP$ over curve
$obj \leftarrow \texttt{enc}(x_r, y_r)$
$tok \leftarrow \perp$
$\vec{c} \leftarrow (y_r^2, y_r^2)$ in $\mathbb{F}^\infty$
return $(obj, tok, \vec{c})$

confirmation code

<u>VrfyC(vk, pk, obj, tok):</u>

parse $(x_r, y_r) \leftarrow \texttt{dec}(obj)$
compute $x_r^3 + ax_r + b$ and y_r^2 over $\mathbb{F}^\infty$
$d \leftarrow [x_r^3 + ax_r + b = y_r^2$ over $\mathbb{F}^\infty]$
$\vec{c} \leftarrow (x_r^3 + ax_r + b, y_r^2)$
return $(d, \vec{c})$

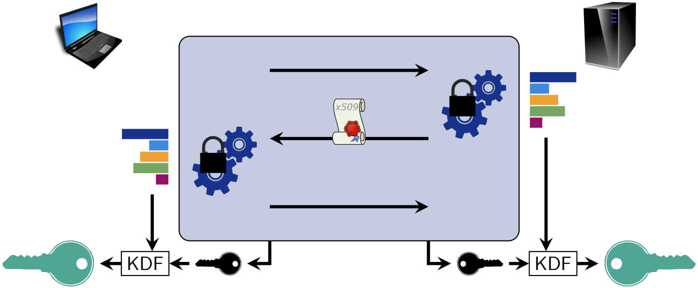
confirmation code

- **unpredictable**: because it's hard to hardcode coordinates of random curve point...
  - note: you need both sides of the check equation in the confirmation code

Making Cryptographic Protocols Fail Noticeably

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

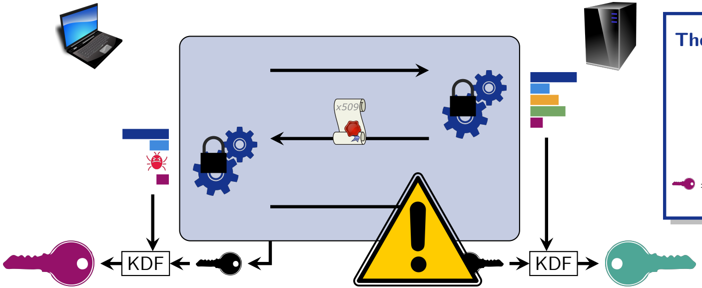
- Better: use confirmation codes in overall protocol — here: **secure connection establishment**



Making Cryptographic Protocols Fail Noticeably

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

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



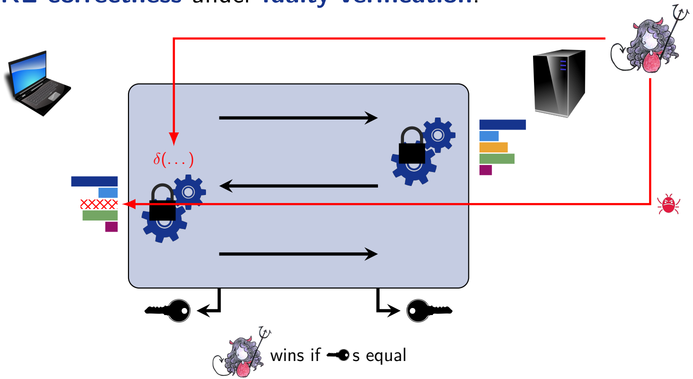
Theorem:

unpredictable
+
KDF collision-resistant
⇓
≠ **noticeably incorrect** under **faulty verification**

Making Cryptographic Protocols Fail Noticeably

Noticeable (In)Correctness under Faulty Verification

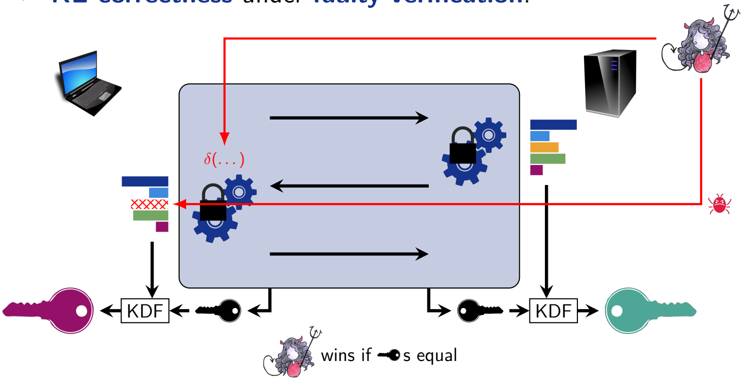
- **KE correctness**: run protocol between A and B, then $K_A = K_B$
- **KE correctness** under **faulty verification**:



Making Cryptographic Protocols Fail Noticeably

Noticeable (In)Correctness under Faulty Verification

- **KE correctness**: run protocol between A and B, then $K_A = K_B$
- **KE correctness** under **faulty verification**:



Theorem (correctness):

unpredictable
+
KDF collision-resistant
$\Downarrow$
$\neq$ **noticeably incorrect** under **faulty verification***

* for "canonical" KE

Theorem (BR security):

KE BR-secure
+
KDF PRF-secure
$\Downarrow$
KE' BR-secure*

* for publicly computable

Discussion

Further Directions & Deployment

- 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
- **Deployment** of confirmation codes requires system-level efforts (beyond the scope of this work)
  - **API**: might surface confirmation codes optionally, for backwards compatibility
  - **Live vs static**: confirmation codes for online signatures are produced live – how best integrate confirmation codes of static signatures (e.g., in certificates)
  - **Transient**: should one be able to (de)activate the usage of confirmation codes? (think: GREASE)

Summary

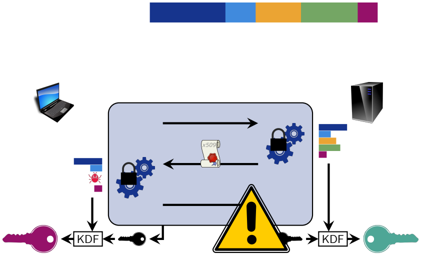
We

- 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**,
  in primitives, protocols, and beyond verification



Thank You!

mail@**felixguenther.info**

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