Short Double- and N-times-Authentication-Preventing Signatures from ECDSA and More

David Derler‡, Sebastian Ramacher‡, Daniel Slamanig§
EUROSP’18, April 25, 2018
Motivation
Digital Signatures

Signer (🔑, 📄) ➔ Verifier (🔍)

📄 ← Sign(🔑, 📄)

Verify(🔍, 📄) = 1✓

Applications

- Signing transactions in cryptocurrencies
- Certificate and software signing
- And many more
Penalize Double-Spending

customer

merchants
Penalize Double-Spending

create offline payment channel

customer

deposit BTC

merchants
Penalize Double-Spending

- customer
- merchants

customer → merchants

transaction 1

deposit /btc

create offline payment channel

/one.osf

/two.osf

/three.osf

receive deposit on misuse

/one.osf

/two.osf

/three.osf
Penalize Double-Spending

customer

transaction 2

merchants

deposit/btc

create offline payment channel

three.osf

receive deposit on misuse
Penalize Double-Spending

customer

벽

merchants

deposit on misuse

create offline payment channel

transaction

three.osf

transaction

one.osf

transaction

two.osf
Penalize Double-Spending

(customer) → lock → (merchants) → receive deposit on misuse
• Same context, different content
  » Can extract secret key
• Extraction from *honest* and *malicious* keys
Existing DAPS

Existing schemes

- Factoring based [PS14, PS17, BPS17]
- DLOG based [RKS15]
- All of them based on trapdoor properties
Existing DAPS

Existing schemes

- Factoring based [PS14, PS17, BPS17]
- DLOG based [RKS15]
- All of them based on trapdoor properties

Problems:

- Factoring based: not compatible with plain RSA signatures
- DLOG based: inefficient
Can we build **efficient** DAPS from existing signature schemes in a **black-box** way?
Black-box Extension

Signature scheme $\Sigma$

Sign(.sec, ...)  Verify(.-,$\Sigma$, ...)
Black-box Extension

DAPS scheme

Signature scheme $\Sigma$

Sign($\mathcal{K}$-$\Sigma$, ...), Verify($\mathcal{K}$-$\Sigma$, ...)

Sign($\mathcal{K}$, ...), Verify($\mathcal{K}$, ...)
Black-box Extension

DAPS scheme

Signature scheme \( \Sigma \)

\[ \text{Sign}(\text{key}, ...) \]

\[ \text{Verify}(\text{key}, ...) \]

\[ \text{Sign}(\text{key}, ...) \]

\[ \text{Verify}(\text{key}, ...) \]

uses

does not use

\( \text{DAPS secret key contains } \Sigma \text{ secret key} \)
Black-box Extension

DAPS scheme

Signature scheme $\Sigma$

Sign($\Sigma$, ...)

Verify($\Sigma$, ...)

uses

Sign($\Sigma$, ...)

Verify($\Sigma$, ...)

dpi

DAPS secret key contains $\Sigma$ secret key
Extraction of $\Sigma$ secret key often sufficient

✓ Example: ECDSA key protecting Bitcoin deposit
Observations

Extraction of $\Sigma$ secret key often sufficient

- Example: ECDSA key protecting Bitcoin deposit
  - New security notions covering $\Sigma$ secret key extraction
  - for honest and malicious keys
Extraction of $\Sigma$ secret key often sufficient

- Example: ECDSA key protecting Bitcoin deposit
- New security notions covering $\Sigma$ secret key extraction
  - for honest and malicious keys
Extraction of $\Sigma$ secret key often sufficient

✔ Example: ECDSA key protecting Bitcoin deposit

» New security notions covering $\Sigma$ secret key extraction

+ for honest and malicious keys

Most applications

• Polynomial address space sufficient
Construction
Shamir Secret Sharing

\[ f(x) = \rho_a x + s k_{\Sigma} \]
Shamir Secret Sharing

\[ f(x) = \rho a x + s k \sum \]

One point reveals nothing about \( s k \sum \).

Two points allow to recover \( s k \sum \).
Shamir Secret Sharing

\[ f(x) = \rho_a x + sk_\Sigma \]

- One point reveals nothing about \( sk_\Sigma \)
Shamir Secret Sharing

\[ f(x) = \rho_a x + s k_\Sigma \]

- One point reveals nothing about \( s k_\Sigma \)
- Two points allow to recover \( s k_\Sigma \)
Generic DAPS

$sk_\Sigma$

$pk_\Sigma$

signature on
encrypt coefficients
secret sharing of
consistency proof
nine.osf
Generic DAPS

\[ \text{sk}_\Sigma \]

\[ \text{pk}_\Sigma \]

\[ m \quad a \quad p \]

signature on encrypt coefficients
secret sharing of consistency proof

/nine.osf
Generic DAPS

signature on $m$

$$sk_{\Sigma}$$

$$pk_{\Sigma}$$

$m$

$a$ $p$

$\sigma$

$\sigma_{\Sigma}$
Generic DAPS

secret sharing of $sk_\Sigma$

$pk_\Sigma$  
$sk_\Sigma$  
$m$  
$\sigma$  
$\rho_a$  
$\sigma_\Sigma$  
$z$
Generic DAPS

- **$\mathbf{sk}_\Sigma$** and **$\rho a$**
- **$\mathbf{pk}_\Sigma$** and **$\mathbf{pk}_E$** leading to **$C_a$**

- **m**
  - **a** and **p**

- **σ**
  - **$\sigma_\Sigma$** and **z**

- Encrypt coefficients
Generic DAPS

\[ \text{sk}_\Sigma \quad \rho_a \]

\[ \text{pk}_\Sigma \quad \text{pk}_E \quad C_a \]

\[ m \quad a \quad p \]

\[ \sigma \quad \sigma_\Sigma \quad z \quad \pi \]
Generic DAPS

\[ \text{key} \quad sk_\Sigma \quad \rho_a \]

\[ \text{public keys} \quad \text{pk}_\Sigma \quad \text{pk}_E \quad C_a \]

\[ m \quad a \quad p \]

\[ \sigma \quad \sigma_\Sigma \quad z \quad \pi \]

consistency proof
Generic DAPS: Wrap Up

Generic approach:

- Black-box use of $\Sigma$
- Verifiable Shamir secret sharing of $\Sigma$ secret key
- Sharing polynomial determined by address

$$f(x) = \rho_a x + s_k$$
Generic DAPS: Wrap Up

Generic approach:

- **Black-box** use of $\Sigma$
- **Verifiable Shamir secret sharing** of $\Sigma$ secret key
- Sharing polynomial determined by address

$$f(x) = \rho_a x + s \text{k}_\Sigma$$

- Evaluate verification relation in encrypted domain
- Zero-knowledge proof of consistency
Generic DAPS: Wrap Up

Generic approach:

- **Black-box** use of $\Sigma$
- **Verifiable Shamir secret sharing** of $\Sigma$ secret key
- Sharing polynomial determined by address

$$f(x) = \rho a x + sk_\Sigma$$

- Evaluate verification relation in encrypted domain
- Zero-knowledge proof of consistency

Can prove unforgeability via unforgeability of $\Sigma$ (**black-box**)
+ For example, applies to ECDSA, EdDSA, DSA
+ Short DAPS signatures
+ Public key linear in size of address space
  (contains encrypted sharing polynomials per address)
For example, applies to ECDSA, EdDSA, DSA

Short DAPS signatures

Public key linear in size of address space
  (contains encrypted sharing polynomials per address)

Extendable to \( N \)-authentication preventing signatures

Use degree \( N - 1 \) sharing polynomial
Implementation

- Easy extension of existing implementations
  + Implement secret sharing
  + Implement consistency proof
  ✔ We provide implementation in OpenSSL
Implementation

- Easy extension of existing implementations
- Implement secret sharing
- Implement consistency proof

Yes We provide implementation in OpenSSL

| Scheme            | Sign [ms] | Verify [ms] | |sk| [bits] | |pk| [bits] | |σ| [bits] |
|-------------------|-----------|-------------|-----------------|-----------------|-----------------|
| ECDSA-DAPS (s)    | 0.76      | 1.33        | 256 \cdot (1 + 2n) | 514 \cdot (1 + n) | 1280             |
| ECDSA-DAPS (p)    | 0.23      | 0.35        | 256 \cdot (1 + 2n) | 514 \cdot (1 + n) | 1280             |
| ECDSA (s)         | 0.09      | 0.35        | 256              | 257              | 512              |
| ECDSA (p)         | 0.06      | 0.21        | 256              | 257              | 512              |

Table 1: Runtime and sizes; secp256k1 (s), prime256v1 (p)
Conclusion
Conclusion

Contribution

✔ Generic construction
✔ Can extend virtually all DLOG-based signature schemes
✔ Focus on extraction of underlying signature scheme key
✔ Shortest black-box DAPS
  (slightly weaker, yet very reasonable model)
✔ Extendable to $N$-authentication preventing signatures
Conclusion

Contribution

☑️ Generic construction
☑️ Can extend virtually all DLOG-based signature schemes
☑️ Focus on extraction of underlying signature scheme key
☑️ Shortest black-box DAPS
  (slightly weaker, yet very reasonable model)
☑️ Extendable to \( N \)-authentication preventing signatures

Follow-up work [Poe18]

- Even shorter DAPS (non-black-box)
Conclusion

Contribution

- Generic construction
- Can extend virtually all DLOG-based signature schemes
- Focus on extraction of underlying signature scheme key
- Shortest black-box DAPS
  (slightly weaker, yet very reasonable model)
- Extendable to $N$-authentication preventing signatures

Follow-up work

- Even shorter DAPS (non-black-box)

Future work

- Reduce public key overhead per address

[Po18]
Questions?

Implementation: https://github.com/IAIK/daps-dl

Supported by: prisma cloud


