Double-Authentication-Preventing Signatures in the Standard Model

Dario Catalano[1] Georg Fuchsbauer [2] Azam Soleimanian[3,4]

[1]Dipartimento di Matematica e Informatica – Università di Catania, Italy
catalano@dmi.unict.it

[2]TU Wien, Vienna, Austria

[3]Inria de Paris, France

[4]École normale supérieure, CNRS, PSL University, Paris, France

{georg.fuchsbauer,azam.soleimanian}@ens.fr

What is DAPS?

- It is a Signature Scheme....

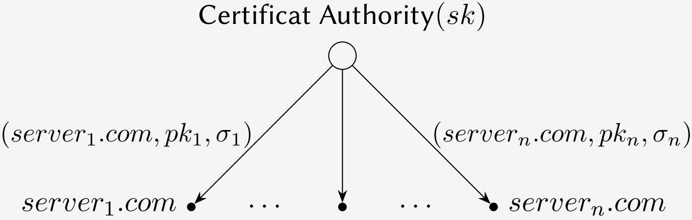
Signer/Authority		Verifier
$(sk, vk) \leftarrow$ KeyGen(1^κ)		
	$\xrightarrow{vk}$	
Signer		**Verifier**
$\sigma_i \leftarrow$ Sign(sk, m_i)		
	$\xrightarrow{(m_i,\sigma_i)}$	
		$1/0 \leftarrow$ Verif$(vk, (m_i, \sigma_i))$

What is DAPS?

- It is a Signature Scheme
- The signer is restricted!

Applications

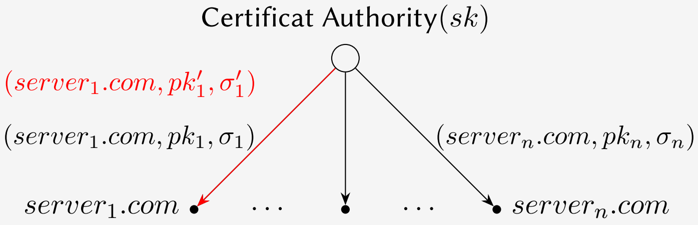
- Certificate subversion
- Cryptocurrencies



$$\sigma_i \leftarrow \mathsf{Sign}_{sk}(server_i.com, pk_i)$$

Applications

- Certificate subversion
- Cryptocurrencies



$$\sigma_i \leftarrow \mathsf{Sign}_{sk}(server_i.com, pk_i)$$

Applications

Applications

- Certificate subversion
- Cryptocurrencies and non-equivocation contracts

$$TX : \mathsf{Sign}_{sk}(coin, reciever) \implies \text{integrity + undeniability}$$

$$\sigma_i \leftarrow \mathsf{Sign}_{sk_i}(coin, reciever)$$

$$\sigma_i \leftarrow \mathsf{Sign}_{sk_i}(coin, reciever)$$

Double-Spending: The same coin for two different receivers

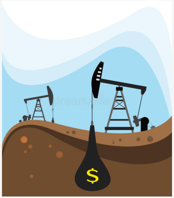
What is DAPS?

It is a Signature Scheme with messages of the form $m = (a, p)$ and equipped with a self-enforcement mechanism.

$$\text{If} \quad \begin{cases} m_1 = (a, p_1) \\ m_2 = (a, p_2) \\ p_1 \neq p_2 \\ \mathsf{Verif}(m_1, \sigma_1) = 1 \\ \mathsf{Verif}(m_2, \sigma_2) = 1 \end{cases} \xrightarrow{\text{Then}} \text{Penalize the Signer}$$

(extract info about sk)

Compromising pair



How It Helps?

- Certificate subversion
- Cryptocurrencies: Blockchain with off-chain payments

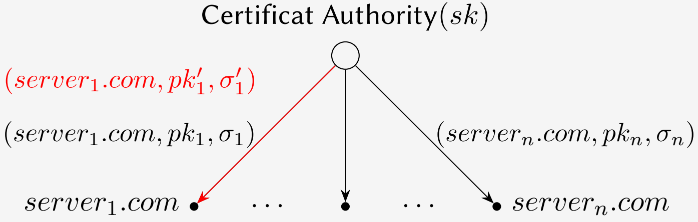
(time-locked) Deposit

w_1 $\mathsf{lock}_{sk_1}(w_1)$ $\ldots$ w_n $\mathsf{lock}_{sk_n}(w_n)$

- Retrieve: $\mathsf{Sign}_{sk_i}(w_i, reciver_i), \quad t > T_{protocol} + T_{claim}$
- Penalize: $\mathsf{Sign}_{sk_i}(w_i, reciver_j), \quad j \neq i$

How It Helps?

- Certificate subversion
- Cryptocurrencies



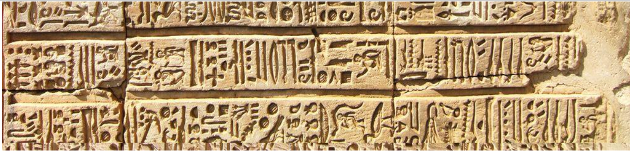
$$\sigma_i \leftarrow \mathsf{Sign}_{sk}(server_i.com, pk_i)$$

Challenges and Contributions

- Exponentially large address space
- Security without trusted setup
- Standard assumptions
- A more general definition
- Concrete instantiation

Related Work

- In ROM
- Small address space
- Trusted setup



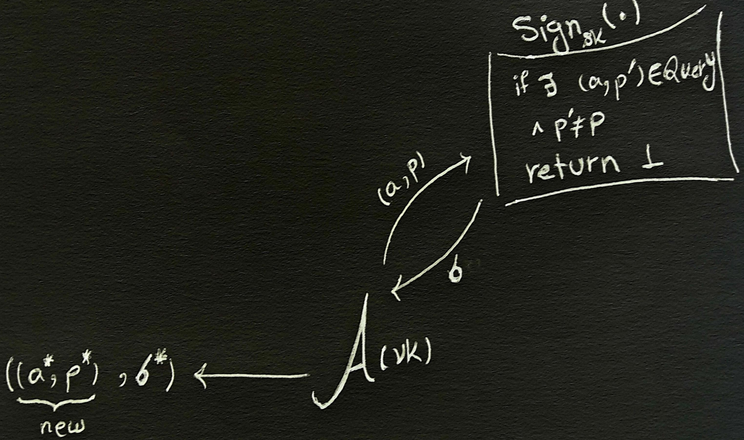
Scheme	Signature size	vk size	Address space	Assumption	ROM	No trusted setup
[Poe18]	$\lvert\mathbb{G}\rvert$	$O(2^n)$	poly.	DLog	yes	no
[RKS15]	$q \cdot h \cdot \lvert\mathbb{G}\rvert$	$O(1)$	exp.	DLog	yes	yes
[PS14]	$(\lambda_H + 1) \cdot \log N$	$O(1)$	exp.	Fact	yes	no
[BPS17]	$\log N$	$O(1)$	exp.	Fact	yes	no
[BKN17]	$O(n_0^2 \log q_0)$	$O(n_0^4 \log^3 q_0)$	exp.	LWE/SIS	yes	yes
[DRS18b]	$\ell_\pi(n)$	$O(2^n)$	poly.	DLog	yes	yes
[LGW+19]	$\log N$ or $2 \cdot \lvert\mathbb{G}\rvert$	$O(1)$	exp.	Fact or CDH	yes	yes
[DRS18a]	$\ell_\pi(n)$	$O(1)$	exp.	PRF & OWF	yes	yes
DAPS-GS	$36n \cdot \lvert\mathbb{G}\rvert$	$O(1)$	exp.	SXDH	no	no
DAPS-VC-DCH	$3h \cdot \lvert\mathbb{G}\rvert$	q	exp.	CDH	no	no
DAPS-DCH	$q \cdot h \cdot \lvert\mathbb{G}\rvert$	$O(1)$	exp.	DLog	no	yes

Syntax and Security:

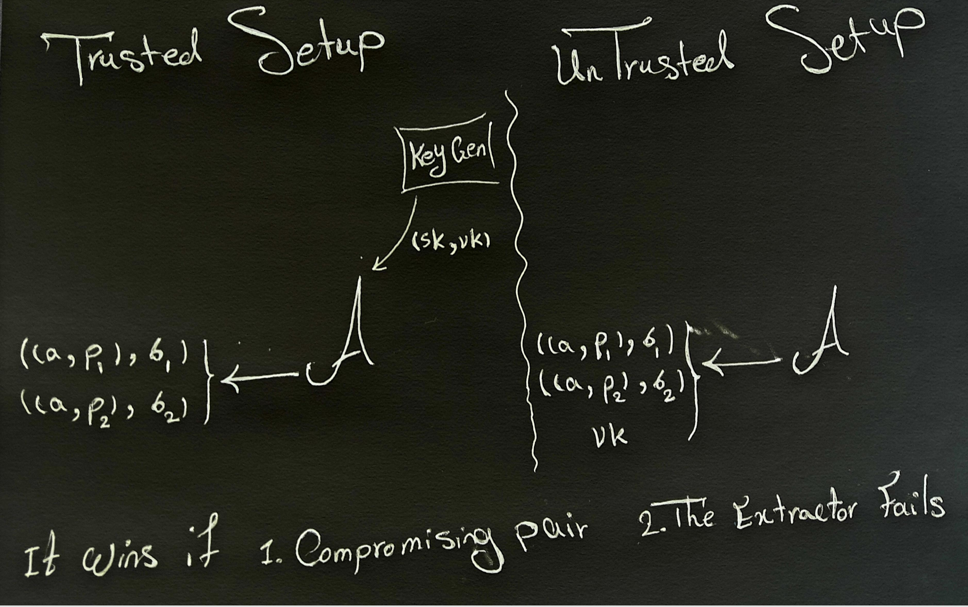
Lets talk more technically...

- Syntactically:
  - $(sk, vk) \leftarrow \mathsf{KeyGen}(1^\kappa)$
  - $\sigma \leftarrow \mathsf{Sign}_{sk}((a, p))$
  - $0/1 \leftarrow \mathsf{Verif}(vk, (a, p), \sigma)$
  - $sk' \leftarrow \mathsf{Ext}(vk, (a, p_1, \sigma_1), (a, p_2, \sigma_2))$
- Security:
  - Unforgeability (outside attacker)
  - Key-Extractability (malicious signer)

Unforgeability:



Key Extractability:



Building Blocks:

- Vector Commitment (VC)
- Double Trapdoor Chameleon hash Function(DCH)

Building Blocks:

- Vector Commitment (VC)
- Double Trapdoor Chameleon hash Function (DCH)

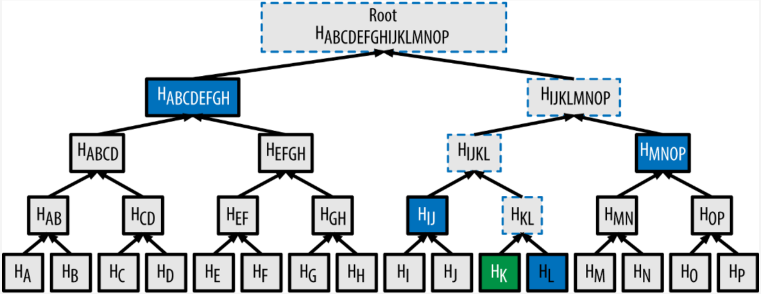
VC: Commit to an ordered sequence of q values. Later open the commitment at a specific position.

- Security

  Position binding: Attacker tries to open the same commitment to two different values in position i.

Vector Commitment

- Merkle Tree is a VC scheme with opening-size $\log n$.
- Can we have a VC with constant-size of opening? $\longrightarrow$ (crs+paring)



Building Blocks:

- Vector Commitment (VC)
- Double Trapdoor Chameleon hash Function (DCH)

DCH: A collision-resistant (CR) hash function with double trapdoors, where given the trapdoor one can find collisions efficiently.

• Security

CR: given one of the trapdoor, is hard to find the other trapdoor

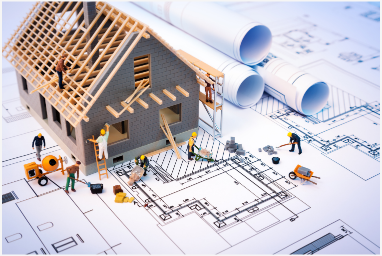
KE: a collision pair leads to revealing of one of the trapdoors

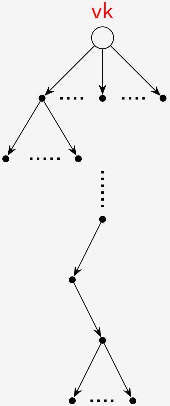
Distribution of collisions: output of Coll seems uniform.

Construction

Big Picture:

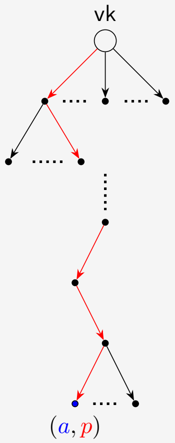
- flat-tree structure
- root-value is fixed as the verification key





Big Picture:

- flat-tree structure
- root-value is fixed as the verification key
- address a is (the position of) the leaf
- the path to the root is weighted by values (depending on p)
- σ: the concatenation of all the values in the path



- Exponential Address-Space.

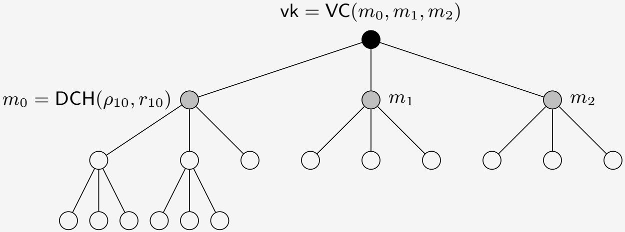


Figure: Generating the verification key

VC instead of CR hash function, shorter signature.

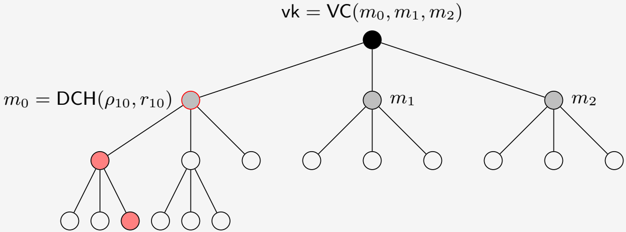
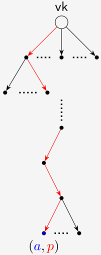


Figure: Signing



When you arrive to a visited node, connect it to the path by finding a collision for DCH.

Why It Is Secure?

The root is fixed with $vk \implies$ **Collision Point on the path**

not (KE of DAPS) $\implies$ not (KE of DCH $\wedge$ position-binding of VC)

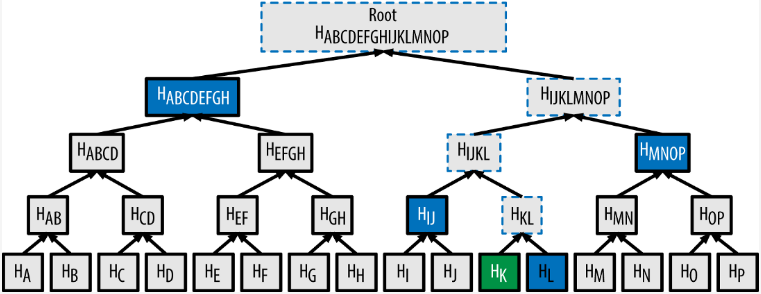
not (Unforg of DAPS) $\implies$ not (CR of DCH $\wedge$ position-binding of VC)

Instantiation

- Vector Commitment (Catalano-Fiore VC scheme [CF13], CDH Ass.)
- Double Trapdoor Chameleon Hash (Our DCH scheme)

Vector Commitment

- Merkle Tree is a VC scheme with opening-size $\log n$.
- Can we have a VC with constant-size of opening? $\longrightarrow$ (crs+paring)



Vector Commitment

KeyGen$(1^\kappa, q)$: select two groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order p equipped with a bilinear map $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$. Let $g \in \mathbb{G}$ be a random generator.

– sample $z_i, \ldots, z_q \xleftarrow{R} \mathbb{Z}_p$.

– set $h_i = g^{z_i}$ for $i = 1, \ldots, q$ and $h_{ij} = g^{z_i z_j}$ for $i, j = 1, \ldots, q$ and $i \neq j$.

Return $\mathsf{pp} = (g, \{h_i\}_i, \{h_{ij}\}_{i,j})$ and define $\mathcal{M} = \mathbb{Z}_p$

Cmt$_{\mathsf{pp}}(m_1, \ldots, m_q)$: compute $C = h_1^{m_1} h_2^{m_2} \ldots h_q^{m_q}$ (where $m_i \in \mathbb{Z}_p$).
Return C.

Open$_{\mathsf{pp}}(m_i, i, m)$: compute $\Lambda_i = \prod_{j=1, j\neq i}^{q} h_{i,j}^{m_j} = \left(\prod_{j=1, j\neq i}^{q} h_j^{m_j}\right)^{z_i}$ return Λ_i.

Verif(C, m_i, i, Λ_i): Output 1 iff $e(C/h_i^{m_i}, h_i) = e(\Lambda_i, g)$

Fig. 11. Catalano-Fiore VC scheme [9]

- Aggregatable $\longrightarrow$ Dec 2020 (PointProof [Gorbunov,Wee,...])
- Updatable
- Short CRS

Double Trapdoor Chameleon Hash

- KeyGen(1^κ): output $tk = (tk_0, tk_1) \xleftarrow{R} \mathbf{Z}_p, pk_0 = g^{tk_0},\ pk_1 = g^{tk_1}$.
- CHash(m, r, s): output $h = g^m \cdot pk_0^r \cdot pk_1^s$
- Coll(tk_i, m, r, s): if tk_0 is given then it is enough to set $s = s'$.
- Ext($(m, r, s), (m', r', s')$): Error!

For a Collision:

$$m + r \cdot tk_0 + s \cdot tk_1 = m' + r' \cdot tk_0 + s' \cdot tk_1$$

Our DCH:

Underlying idea: One equation, one unknown!

Let $\mathcal{H}_0$ and $\mathcal{H}_1$ be Chameleon hash functions.

$$C \leftarrow \mathsf{CHash}_{\mathsf{pk}}(m, r, s) \text{ where } \begin{cases} w = \mathcal{H}_0.\mathsf{CHash}(m, r) \\ C = \mathcal{H}_1.\mathsf{CHash}(w, s) \end{cases}$$

- Instantiation based on **DLog**.

DAPS in Untrusted Setup?

- Our DCH scheme is Secure against Untrusted Setup.
- There is no VC scheme Secure against Untrusted Setup!

Q: How we can get a DAPS scheme secure in Untrusted Setup?
A: Replace VC with a standard CR Hash Function (with the cost of longer signature).

Open Questions

- Constant-size DAPS-signature in the standard model (ours is of size $\log_q n$)
- Is it possible to have a (constant-size) VC scheme secure against untrusted setup?
- Smart Contract from DAPS for different applications.