Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically

Simulation extractable, subversion, and updatable NIZKs

Sebastian Ramacher, joint work with Behzad Abdolmaleki and Daniel Slamanig

March 4, 2020

AIT Austrian Institute of Technology, Vienna
Introduction
Zero-knowledge Proofs

**NP-language** $L$

- Prover wants to convince verifier that some $x \in L$
- Without revealing information beyond the statement $x \in L$
- Define relation $R_L: x \in L \iff \exists w: (x, w) \in R_L$

![Diagram showing interaction between Prover and Verifier](image-url)
Making them Non-Interactive: ROM

Random-oracle model: Fiat-Shamir transform [FS86], Unruh transform [Unr15]

\[ \pi \leftarrow \text{Prove}_H(x, w) \]

\[ \text{Verify}_H(x, \pi) = 1 \Rightarrow x \in L \]
Common reference string model

\[ \pi \leftarrow \text{Prove}(\text{crs}, x, w) \]

\[ \text{Verify}(\text{crs}, x, \pi) = 1 \Rightarrow x \in L \]
Important Properties

Prover cannot cheat

- Prover unable to produce valid proofs for $x \notin L$

> Soundness

- Property desired by the verifier
Important Properties

Prover cannot cheat

- Prover unable to produce valid proofs for $x \not\in L$

  > Soundness

- Property desired by the verifier

Verifier does not learn any information on witness $w$

- Real proofs cannot be distinguished from simulated proofs

  > Zero-knowledge

- Property desired by the prover
Important Properties

Proofs of Knowledge

- Special extractor can extract witness from proofs
  - Knowledge Soundness
Important Properties

Proofs of Knowledge

- Special extractor can extract witness from proofs
  - Knowledge Soundness

Strong versions

- (Knowledge) Soundness also holds if adversary can query simulated proofs
  - Simulation (knowledge) soundness
- SKS also called simulation (sound) extractability (SE)
NIZKs in the CRS Model

- Zero-knowledge contradicts extractor
- Soundness contradicts simulator
NIZKs in the CRS Model

- Zero-knowledge contradicts extractor
- Soundness contradicts simulator

They need to have more power

- Extractor gets extraction trapdoor
- Simulator gets simulation trapdoor
NIZKs in the CRS Model

TTP crs

Prover \((x, w)\)

\(\pi \leftarrow \text{Prove}(\text{crs}, x, w)\)

Verifier \(x\)

\(\text{Verify}(\text{crs}, x, \pi) = 1 \Rightarrow x \in L\)

Extractor

Simulator

\(\text{Simulator} \)

\(\text{Extractor} \)

\(\pi \leftarrow \text{Prove}(\text{crs}, x, w)\)
NIZKs in the CRS Model

\[ \pi \leftarrow \text{Prove}(\text{crs}, x, w) \]

\[ \text{Verify}(\text{crs}, x, \pi) = 1 \Rightarrow x \in L \]
NIZKs in the CRS Model

\[ \text{TTP} \quad \text{crs}, t_s, t_e \]

\[ \pi \leftarrow \text{Prove}(\text{crs}, x, w) \]

\[ \text{Verifier} \quad x \]

\[ \text{Verify}(\text{crs}, x, \pi) = 1 \Rightarrow x \in L \]

\[ w \leftarrow \text{Ext}(\text{crs}, t_e, \pi) \]

\[ \pi \leftarrow \text{Sim}(\text{crs}, t_s, x) \]
Achieving Simulation Extractability
Folklore compiler [SP92]: “$x \in L$ and I have encrypted $w$”

- $\Omega$: perfectly correct IND-CPA public-key encryption
- Extend CRS with a public key of $\Omega$: $\text{pk}_\Omega$
- Extend proof with encryption of $w$: $c = \Omega\cdot\text{Enc}(\text{pk}_\Omega, w; r)$
- Extend statement to $(x, w) \in R \land c = \Omega\cdot\text{Enc}(\text{pk}_\Omega, w; r)$
- Put secret key in extraction trapdoor $t$
Folklore compiler [SP92]: “$x \in L$ and I have encrypted $w$”

- $\Omega$: perfectly correct IND-CPA public-key encryption
- Extend CRS with a public key of $\Omega$: $pk_\Omega$
- Extend proof with encryption of $w$: $c = \Omega.\text{Enc}(pk_\Omega, w; r)$
Folklore compiler [SP92]: “$x \in L$ and I have encrypted $w$”

- $\Omega$: perfectly correct IND-CPA public-key encryption
- Extend CRS with a public key of $\Omega$: $pk_{\Omega}$
- Extend proof with encryption of $w$: $c = \Omega.\text{Enc}(pk_{\Omega}, w; r)$
- Extend statement to

$$(x, w) \in R_L \land c = \Omega.\text{Enc}(pk_{\Omega}, w; r)$$
Folklore compiler [SP92]: “$x \in L$ and I have encrypted $w$”

- $\Omega$: perfectly correct IND-CPA public-key encryption
- Extend CRS with a public key of $\Omega$: $pk_\Omega$
- Extend proof with encryption of $w$: $c = \Omega.\text{Enc}(pk_\Omega, w; r)$
- Extend statement to

\[(x, w) \in R_L \land c = \Omega.\text{Enc}(pk_\Omega, w; r)\]

- Put secret key in extraction trapdoor $t_e$
In a real world protocol:

- Adversary sees many different proofs

- Might be possible to turn proof $\pi$ for word $x$ into a proof $\pi' \neq \pi$

- Or worse: turn into a proof $\pi'$ for a different word $x' \neq x$
On Simulation Soundness

In a real world protocol:

- Adversary sees many different proofs
- Might be possible to turn proof $\pi$ for word $x$ into a proof $\pi' \neq \pi$
- Or worse: turn into a proof $\pi'$ for a different word $x' \neq x$

Hence

- Adversary may query proofs
- Must produce a proof not queried before

Similar issue for signatures: one-time EUF-CMA – EUF-CMA – strong EUF-CMA
Folklore compiler [GMY03; Gro06]: “$x \in L$ or I know a signature under a public key in the CRS”

- $\Sigma$: EUF-CMA signature scheme
- $\Sigma^1$: strong one-time signature scheme
- Extend CRS with a public key of $\Sigma$: $pk_{\Sigma}$
Folklore compiler [GMY03; Gro06]: “\( x \in L \) or I know a signature under a public key in the CRS”

- \( \Sigma \): EUF-CMA signature scheme
- \( \Sigma^1 \): strong one-time signature scheme
- Extend CRS with a public key of \( \Sigma \): \( pk_{\Sigma} \)
- For a proof
  - Generate new key pair for \( \Sigma^1 \)
Soundness to Simulation Soundness

Folklore compiler [GMY03; Gro06]: “\(x \in L\) or I know a signature under a public key in the CRS”

- \(\Sigma\): EUF-CMA signature scheme
- \(\Sigma^1\): strong one-time signature scheme
- Extend CRS with a public key of \(\Sigma\): \(pk_\Sigma\)
- For a proof
  - Generate new key pair for \(\Sigma^1\)
  - Extend statement to

\[
(x, w) \in R_L \lor \Sigma.\text{Verify}(pk_\Sigma, pk_{\Sigma^1}, \sigma) = 1
\]
Soundness to Simulation Soundness

Folklore compiler [GMY03; Gro06]: “x ∈ L or I know a signature under a public key in the CRS”

- $\Sigma$: EUF-CMA signature scheme
- $\Sigma^1$: strong one-time signature scheme
- Extend CRS with a public key of $\Sigma$: $pk_{\Sigma}$
- For a proof
  - Generate new key pair for $\Sigma^1$
  - Extend statement to
    $$(x, w) \in R_L \lor \Sigma.\text{Verify}(pk_{\Sigma}, pk_{\Sigma^1}, \sigma) = 1$$
  - Sign proof with $sk_{\Sigma^1}$
Soundness to Simulation Soundness

Folklore compiler [GMY03; Gro06]: “\( x \in L \) or I know a signature under a public key in the CRS”

- \( \Sigma \): EUF-CMA signature scheme
- \( \Sigma^1 \): strong one-time signature scheme
- Extend CRS with a public key of \( \Sigma \): \( \text{pk}_\Sigma \)
- For a proof
  - Generate new key pair for \( \Sigma^1 \)
  - Extend statement to
    \[
    (x, w) \in R_L \lor \Sigma.\text{Verify}(\text{pk}_\Sigma, \text{pk}_{\Sigma^1}, \sigma) = 1
    \]
  - Sign proof with \( \text{sk}_{\Sigma^1} \)
  - Put secret key of \( \Sigma \) in simulation trapdoor \( t_s \)
Extend statement to

\[ c = \Omega \cdot \text{Enc}(pk_\Omega, w; r_1) \land ((x, w) \in R_L \lor (\mu = f_\Sigma(pk_{\Sigma^1}) \land \rho = \text{Commit}(s; r_\Omega))) \]

and sign \((x, c, \mu, \pi_{L'})\) with \(sk_{\Sigma^1}\)

\text{crs} extended with \(\rho, pk_\Omega, s, r_0\) simulation trapdorr, \(sk_\Omega\) extraction trapdoor

- \(\Omega\): public-key encryption
- \(\Sigma^1\): strong one-time signature
- \(f\): PRF
- \textbf{Commit}: Commitment
• $\Omega$: RSA
• $\Sigma^1$: RSA-PSS
Instantation of $C\emptyset C\emptyset$

- $\Omega$: RSA
- $\Sigma^1$: RSA-PSS

Not optimal; better alternatives: Boneh-Boyen [BB04], Groth sOTS
Instantation of $\mathcal{C}_{0\mathcal{C}_0}$

- $\Omega$: RSA
- $\Sigma^1$: RSA-PSS
  
  Not optimal; better alternatives: Boneh-Boyen [BB04], Groth sOTS
- $f$: SHA256
- Commit: SHA256
Instantiation of $C_{\emptyset}C_{\emptyset}$

- $\Omega$: RSA
- $\Sigma^1$: RSA-PSS
  Not optimal; better alternatives: Boneh-Boyen [BB04], Groth sOTS
- $f$: SHA256
- **Commit**: SHA256
  Proving pre-image of a random oracle
• \(\Omega\): RSA
• \(\Sigma^1\): RSA-PSS
  Not optimal; better alternatives: Boneh-Boyen [BB04], Groth sOTS
• \(f\): SHA256
• **Commit**: SHA256
  Proving pre-image of a *random oracle*

How to commit to the PRF key while retaining provable security?
Fixed-value key-binding PRF [CMR98; Fis99]

- For a PRF $f$ with key $s$ and special value $\beta$, hard to find $s'$ with $f_s(\beta) = f_{s'}(\beta)$
Fixed-value key-binding PRF [CMR98; Fis99]

- For a PRF $f$ with key $s$ and special value $\beta$, hard to find $s'$ with $f_s(\beta) = f_{s'}(\beta)$

Change statement to

$$(x, w) \in R_L \lor (\mu = f_s(pk_{\Sigma_1}) \land \rho = f_s(\beta))$$
## Instantiation of OC∅C∅

<table>
<thead>
<tr>
<th>Framework</th>
<th>Symmetric primitive</th>
<th>PRF / Commitment</th>
<th># of constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>C∅C∅</td>
<td>SHA256</td>
<td>HMAC PRF + hash com.</td>
<td>244,992</td>
</tr>
<tr>
<td>OC∅C∅</td>
<td>SHA256</td>
<td>HMAC PRF</td>
<td>222,720</td>
</tr>
<tr>
<td></td>
<td>SHAKE256</td>
<td>Sponge PRF</td>
<td>76,800</td>
</tr>
<tr>
<td></td>
<td>MiMC-((1025, 646))</td>
<td>Sponge PRF</td>
<td>1,292</td>
</tr>
<tr>
<td></td>
<td>GMiMC-((1024, 4, 332))</td>
<td>Sponge PRF</td>
<td>1,998</td>
</tr>
<tr>
<td></td>
<td>POSEIDON-((1536, 2, 10, 114))</td>
<td>Sponge PRF</td>
<td>804</td>
</tr>
<tr>
<td></td>
<td>VISION-((1778, 14, 10))</td>
<td>Sponge PRF</td>
<td>2,800</td>
</tr>
<tr>
<td></td>
<td>RESCUE-((1750, 14, 10))</td>
<td>Sponge PRF</td>
<td>1,680</td>
</tr>
<tr>
<td></td>
<td>LOWMC-((1024, 256, 1, 1027))</td>
<td>Sponge PRF</td>
<td>4,288</td>
</tr>
</tbody>
</table>
Beware:

- Numbers from before recent attacks appeared on ePrint
- Numbers are lower bounds assuming PRFs are fixed-value key-binding
- Alternatively: More expensive tree-based construction
Subversion and Updatability
CRS Generator

Prover \((x, w)\)

\[ \pi \leftarrow \text{Prove}(\text{crs}, x, w) \]

Verifier \(x\)

\[ \text{Verify}(\text{crs}, x, \pi) = 1 \Rightarrow x \in L \]
Prover \((x, w)\)

\[\pi \leftarrow \text{Prove}(\text{crs}, x, w)\]

Verifier \(x\)

\[\text{Verify}(\text{crs}, x, \pi) = 1 \Rightarrow x \in L\]
What if the CRS generator is malicious?
Malicious CRS Generator

No guarantee that

- CRS is correct
- CRS from the correct distribution
- Trapdoors exist
Malicious CRS Generator

No guarantee that

- CRS is correct
- CRS from the correct distribution
- Trapdoors exist

Perform CRS generation with MPC protocol

- Examples: zcash ceremony
- But in practice complicated, expensive and requires much effort beside technical realization
Subversion Resistance [BFS16]

- Subversion soundness: sound even if CRS subverted
- Subversion zero-knowledge: zero-knowledge even if CRS subverted
- Some combinations impossible
Subversion Resistance [BFS16]

- Subversion soundness: sound even if CRS subverted
- Subversion zero-knowledge: zero-knowledge even if CRS subverted
- Some combinations impossible

<table>
<thead>
<tr>
<th></th>
<th>WI</th>
<th>Zero-Knowledge</th>
<th>Subversion ZK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soundness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Subversion soundness</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Only ad-hoc constructions so far \([\text{ABL}^*17; \text{Fuc}18; \text{Bag}19]\)

General idea:

- Add public algorithm \(V_{\text{crs}}\)
- If \(V_{\text{crs}}(\text{crs}) = 1\), CRS is valid and simulation trapdoor exists
- In sub-ZK proof, extract trapdoor from CRS for simulation
Updatable NIZK [GKM+18]

- Assume adversary has complete (or partial) control over \( \text{crs} \) generation
- Add \textbf{Ucrs} algorithm: outputs a new CRS and proof of update
- Also add \textbf{Vcrs}: verifies CRS, updates and proofs

Idea: either \( \text{crs} \) was generated honestly or one update was done honestly

- Verifier updates CRS to ensure soundness
- Prover updates CRS to ensure zero-knowledge
Generic framework to obtain

- subversion or updatable
- and simulation extractable zk-SNARKs

Built from

- updatable signatures
- alternative compiler for simulation soundess [DS19]
Key-homomorphic Signatures

• Homomorphism between private-key and public-key spaces: $\mu: S \rightarrow P$
  Natural in the DLOG setting: $x \mapsto g^x$
• Signatures can be adapted from $pk$ to $pk' = pk \cdot \mu(sk' - sk)$ if $sk' - sk$ known
• Examples: Schnorr, BLS, and many more
Simulation Soundness using Key-Homomorphic Signatures

Compiler [DS19]:

- $\Sigma$: key-homomorphic EUF-CMA signature scheme
- $\Sigma^1$: one-time signature scheme
- Extend CRS with a public key of $\Sigma$: $pk$
- For a proof
  - Generate key pairs $(sk', pk')$ for $\Sigma$ and $(sk^1, pk^1)$ for $\Sigma^1$
  - Extend statement to
    $$(x, w) \in R_L \lor pk' = pk \cdot \mu(sk' - sk)$$
  - Sign $pk^1$ with $sk'$ and sign the proof with $sk^1$
  - Put secret key of $\Sigma$ in simulation trapdoor $t_s$

Obtain simulation extractable, subversion zk-SNARK
Updatable Signatures

Similar to updatable CRS

- **Upk**: update \( pk \) and provide proof of update
- **Vpk**: verify update

Idea: either original \( pk \) created honestly or update was done honestly

Example: Schnorr in bilinear groups with BDH knowledge assumption

Obtain simulation extractable, updatable zk-SNARK
## Comparison of SE SNARKs

<table>
<thead>
<tr>
<th></th>
<th>Gen.</th>
<th>Sub.</th>
<th>Upd.</th>
<th>crs</th>
<th>$\pi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CØCØ [KZM⁺15]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>$\kappa$</td>
<td>$2\mathbb{Z}_N, \kappa$</td>
</tr>
<tr>
<td>OCØCØ[G]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>$2\kappa$</td>
<td>$3G, 3\mathbb{Z}_q, \kappa$</td>
</tr>
<tr>
<td>LAMASSU[S,G]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>$G$</td>
<td>$4G, 5\mathbb{Z}_q$</td>
</tr>
<tr>
<td>LAMASSU[S,G]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>$G_1, G_2$</td>
<td>$G_1, G_2, 3G_1, 5\mathbb{Z}_q$</td>
</tr>
<tr>
<td>LAMASSU[S,BB]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>$G$</td>
<td>$G_1, G_2, G_1, G_2, 2\mathbb{Z}_q$</td>
</tr>
<tr>
<td>LAMASSU[S,BB]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>$G_1, G_2$</td>
<td>$2G_1, 2G_2, 2\mathbb{Z}_q$</td>
</tr>
<tr>
<td>Groth-Maller [GM17]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>$(2n + 5)G_1, nG_2$</td>
<td>-</td>
</tr>
<tr>
<td>Bowe-Gabizon [BG18]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>$G_1, G_2$</td>
</tr>
<tr>
<td>Lipmaa (S$q_{se}$) [Lip19]</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>$G_1$</td>
<td>$G_1$</td>
</tr>
<tr>
<td>Kim-Lee-Oh [KLO19]</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>$G_1$</td>
<td>-</td>
</tr>
<tr>
<td>Atapoor-Bagheri [AB19]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>$\kappa$</td>
<td>$G_1, G_2, \kappa$</td>
</tr>
<tr>
<td>Bagheri [Bag19]</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>$\kappa$</td>
<td>$G_1, G_2, \kappa$</td>
</tr>
</tbody>
</table>
Conclusion
Conclusion

C∅C∅, OC∅C∅:

- C∅C∅ hard to instantiate correctly and efficiently
- Even if commitment with enough structure used, C∅C∅ does not seem to yield updatability
- sub-ZK SE SNARK if underlying SNARK already sub-ZK
- OC∅C∅ gives another application of fixed-value key-binding PRFs

LAMASSU:

- generic sub-ZK, updatable SE SNARK
- Open problems: key-homomorphic / updatable signatures from lattices, ...
Questions?


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Conference/Journal</th>
<th>Pages/Volume</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>Title</td>
<td>Conference/Volume</td>
<td>Year</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------------------------</td>
<td>-------------------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Fuc18]</td>
<td>G. Fuchsbauer. Subversion-zero-knowledge snarks.</td>
<td>PKC 10769, LNCS 347</td>
<td>2018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[GKM+18]</td>
<td>J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers.</td>
<td>CRYPTO 10993, LNCS 728</td>
<td>2018</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Venue</th>
<th>URL</th>
</tr>
</thead>
</table>