Benchmarking the Setup of Updatable zk-SNARKs

KARIM BAGHERY, **AXEL MERTENS**, MAHDI SEDAGHAT

KU LEUVEN – COSIC

firstname.lastname@kuleuven.be

Overview

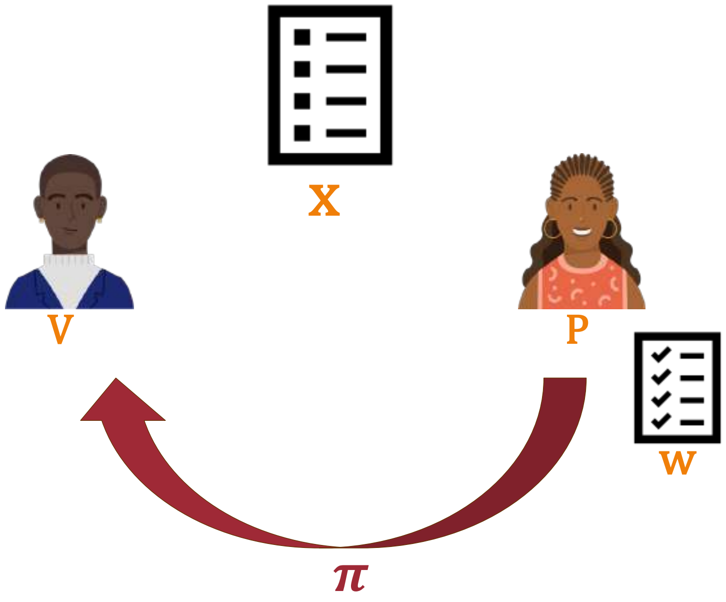
- Preliminaries about zk-SNARKs
- Our Algorithms
- Benchmarks
- Identifiable security

Definitions ZK-proof

- Prover (knowing secret witness) can convince Verifier of a statement x

- Zero Knowledge
  - V gains no information about the witness
- Subversion Zero Knowledge
  - Zero knowledge, also when the srs is subverted

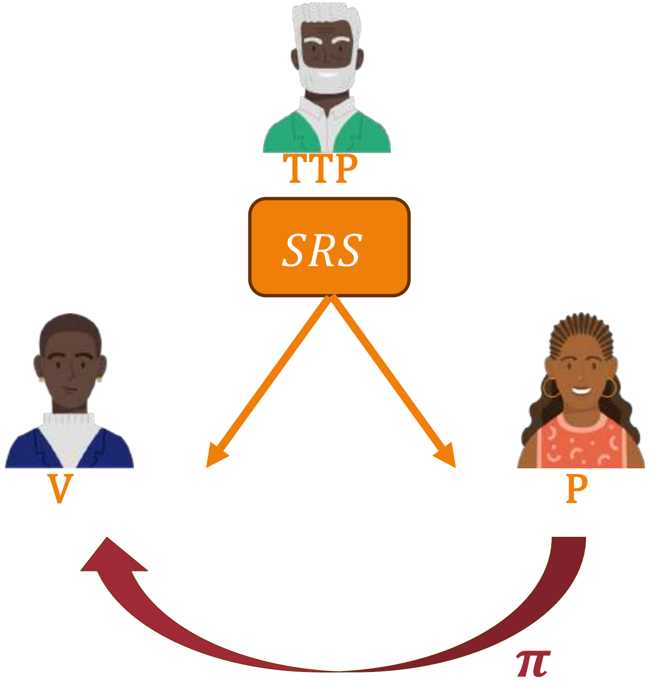
- Knowledge Soundness
  - P cannot convince V without knowing witness
- Updatable Knowledge Soundness
  - Knowledge soundness in the updatable srs setting



What is a zk-SNARK

- Zero-Knowledge Succinct Non-interactive Arguments of Knowledge
- Zero Knowledge
- Knowledge Soundness

- SRS sampled by a trusted third party
- 3 algorithms: (SG, P, V)

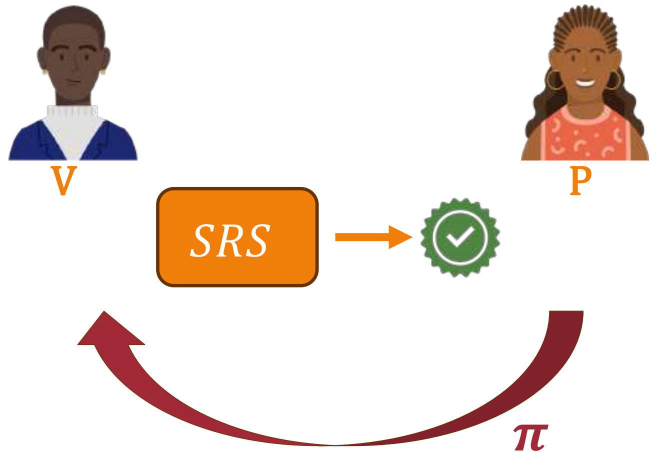


What is a Subversion zk-SNARK

- V generates SRS
- P checks well-formedness of SRS

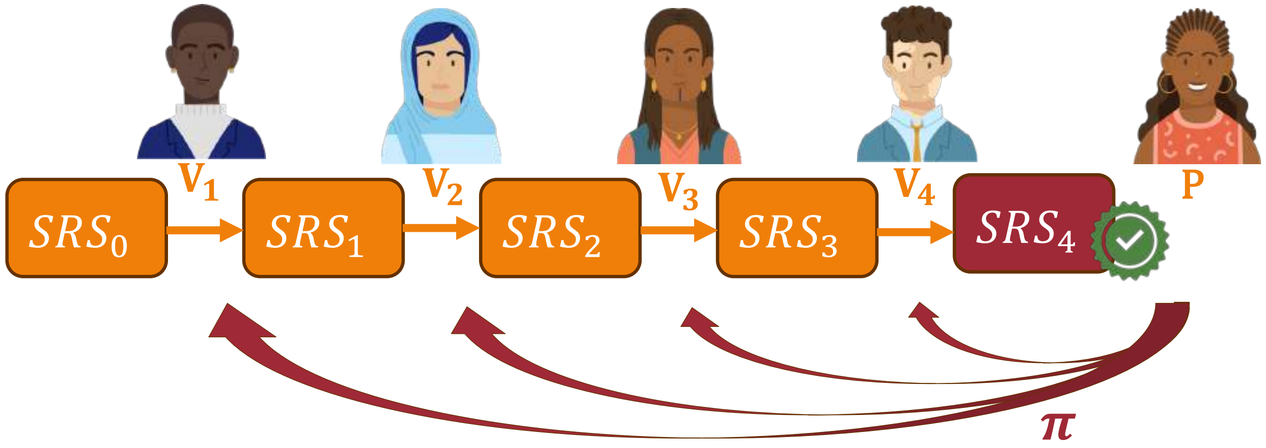
- Multiple Verifiers:
  → MPC protocol: 1-out-of-n

- 4 algorithms: (SG, SV, P, V)



What is an Updatable zk-SNARK

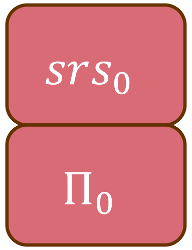
- Distribute the setup
- Updatable Knowledge Soundness:
  - If one update is honest
- Subversion zero knowledge:
  - prover checks the final SRS for well-formedness
- Universal
  - Can be used for any circuit



- 5 algorithms: (SG, SU, SV, P, V)

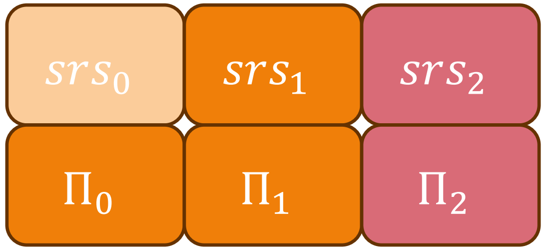
SRS algorithms

- $\mathrm{SG}(d) \rightarrow (srs_0, \Pi_0)$



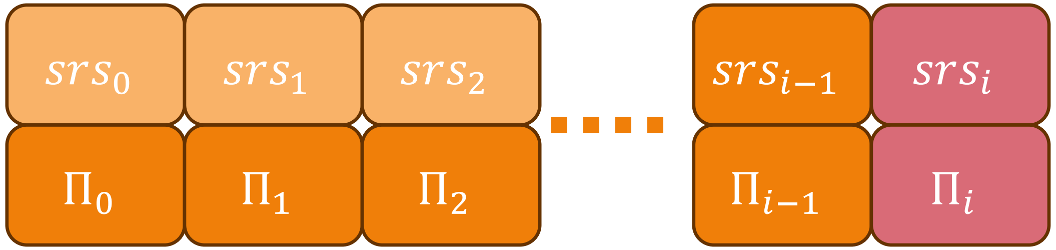
SRS algorithms

- $\mathrm{SG}(d) \rightarrow (srs_0, \Pi_0)$
- $\mathrm{SU}\left(srs_{i-1}, \{\Pi_j\}_{j=0}^{i-1}\right) \rightarrow (srs_i, \Pi_i)$



SRS algorithms

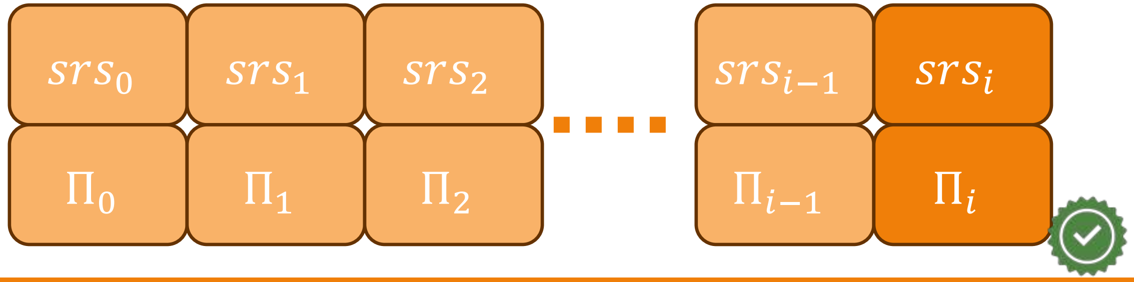
- $\mathrm{SG}(d) \rightarrow (srs_0, \Pi_0)$
- $\mathrm{SU}\left(srs_{i-1}, \{\Pi_j\}_{j=0}^{i-1}\right) \rightarrow (srs_i, \Pi_i)$



SRS algorithms

- $\mathrm{SG}(d) \rightarrow (srs_0, \Pi_0)$
- $\mathrm{SU}\left(srs_{i-1}, \{\Pi_j\}_{j=0}^{i-1}\right) \rightarrow (srs_i, \Pi_i)$
- $\mathrm{SV}\left(srs_i, \{\Pi_j\}_{j=0}^{i}, party\right) \rightarrow \bot/1$

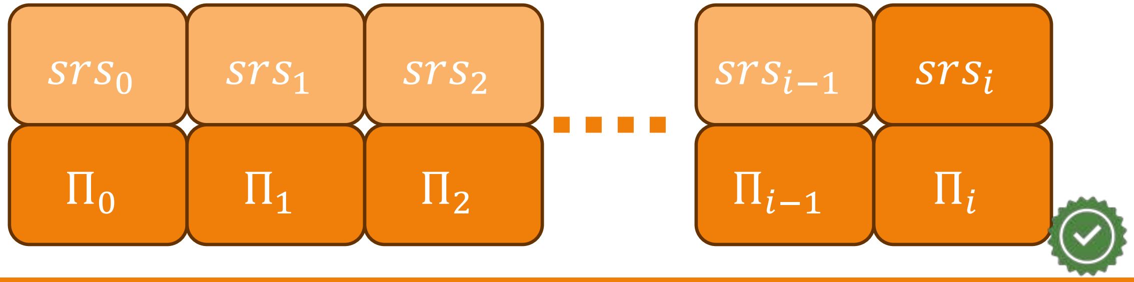
Prover



SRS algorithms

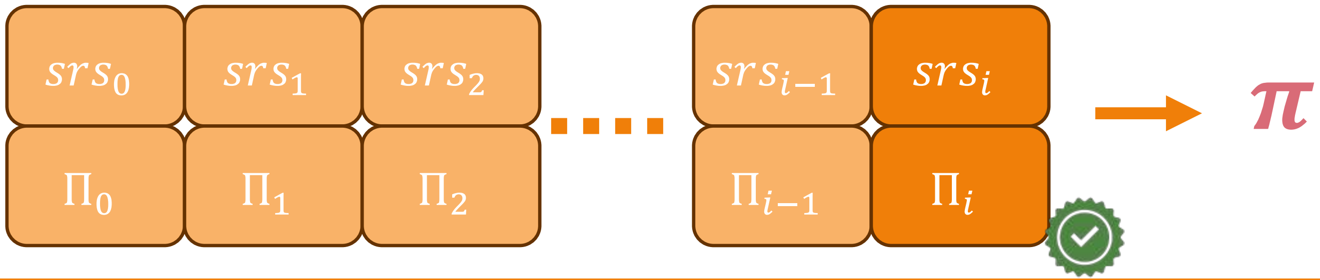
- $\text{SG}(d) \rightarrow (srs_0, \Pi_0)$
- $\text{SU}\left(srs_{i-1}, \{\Pi_j\}_{j=0}^{i-1}\right) \rightarrow (srs_i, \Pi_i)$
- $\text{SV}\left(srs_i, \{\Pi_j\}_{j=0}^{i}, party\right) \rightarrow \perp/1$

Verifier



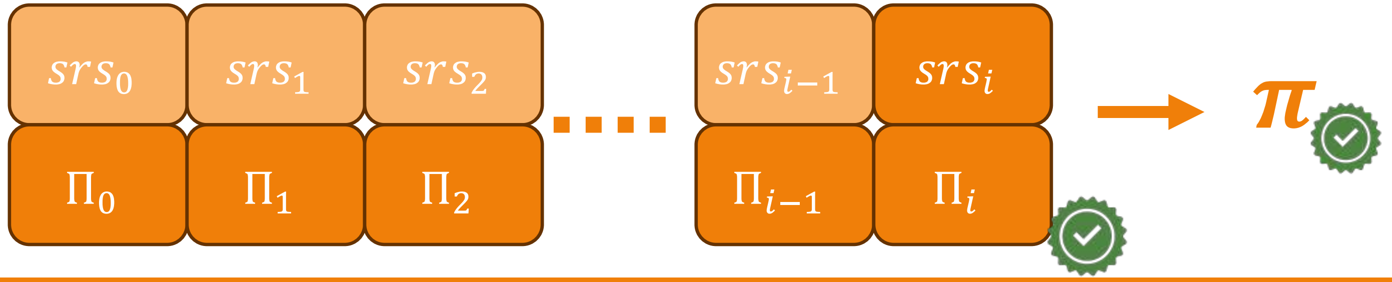
SRS algorithms

- $\mathrm{SG}(d) \ \rightarrow (srs_0, \Pi_0)$
- $\mathrm{SU}\left(srs_{i-1}, \{\Pi_j\}_{j=0}^{i-1}\right) \rightarrow (srs_i, \Pi_i)$
- $\mathrm{SV}\left(srs_i, \{\Pi_j\}_{j=0}^{i}, party\right) \rightarrow \perp/1$
- $\mathrm{P}(R, srs_i, x, w) \rightarrow \perp/\pi$



SRS algorithms

- $\mathrm{SG}(d) \rightarrow (srs_0, \Pi_0)$
- $\mathrm{SU}\left(srs_{i-1}, \{\Pi_j\}_{j=0}^{i-1}\right) \rightarrow (srs_i, \Pi_i)$
- $\mathrm{SV}\left(srs_i, \{\Pi_j\}_{j=0}^{i}, party\right) \rightarrow \bot/1$
- $\mathrm{P}(R, srs_i, x, w) \rightarrow \bot/\pi$
- $V(R, srs_i, x, \pi) \rightarrow 0/1$



Popular Updatable zk-SNARKs

- Sonic
- Plonk
- Marlin
- LunarLite
- Basilisk

-> Pairing-based with the shortest proofs

Popular Updatable zk SNARKS

- Sonic
- Plonk
- Marlin
- LunarLite
- Basilisk
- ~~Counting Vampires~~
  - Shorter π at cost of larger srs

		$\lvert srs \rvert$	$\lvert \pi \rvert$	SG	P	V
Sonic [32]	$\mathbb{G}_1$	$4n-1$	20	$4n-1$	$273n$	$7P$
	$\mathbb{G}_2$	$4n$	–	$4n$	–	
	$\mathbb{F}$	–	16	–	$O(k\log(k))$	$O(m_0+\log(k))$
Plonk [21]	$\mathbb{G}_1$	$3m$	7	$3m$	$11m$	$2P$
	$\mathbb{G}_2$	1	–	–	–	
	$\mathbb{F}$	–	7	–	$O(m\log(m))$	$O(m_0+\log(m))$
Marlin [14]	$\mathbb{G}_1$	$3k$	13	$3k$	$14n+8k$	$2P$
	$\mathbb{G}_2$	2	–	–	–	
	$\mathbb{F}$	–	8	–	$O(k\log(k))$	$O(m_0+\log(k))$
LunarLite [13]	$\mathbb{G}_1$	k	10	k	$8n+3k$	$7P$
	$\mathbb{G}_2$	k	–	k	–	
	$\mathbb{F}$	–	2	–	$O(k\log(k))$	$O(m_0+\log(k))$
Basilisk [35]	$\mathbb{G}_1$	n	6	n	$6n$	$2P$
	$\mathbb{G}_2$	1	–	–	–	
	$\mathbb{F}$	–	2	–	$O(n\log(n))$	$O(m_0+\log(n))$
Vampires [31]	$\mathbb{G}_1$	$12n+k$	4	$12n+k$	$20n+2k$	$5\mathbb{G}_1+6P$
	$\mathbb{G}_2$	$4n+k$	–	$4n+k$	–	21
	$\mathbb{F}$	–	2	–	$O(k\log(k))$	$O(m_0+\log(n))$

Our contributions

- Setup Algorithms for 5 popular Updatable zk-SNARKs
  - Implementation and practical comparison
- Different SV algorithms for P and V
- Batched versions of SV (BSV)
- Altered Marlin SRS
  - AGM does not cover attacks like *hash-to-curve*
- Identifiable security in updatable SRS model

Our general strategy

- Subversion Zero-Knowledge
  - SV that checks well-formedness final SRS
- Updatable Knowledge Soundness
  - SV that checks the correctness of intermediate proofs and final SRS
- Split Π into
  - Π^{Agg} : Aggregated elements for well-formedness
  - Π^{Ind} : Individual proof for each update (not needed for P)
- Batched verification to improve efficiency

Maths Notation

- Additive bracket notation:

- in group $\mathbb{G}_\zeta$, (for $\zeta \in \{1,2,T\}$):

$$[a]_\zeta = a[1]_\zeta$$

- Bilinear pairing:

$$\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T : [a]_1 \bullet [b]_2 = [ab]_T$$

Plonk SRS Generation (SG)

- For $x_0 \leftarrow \mathbb{Z}_p^*$
- $srs_0 := \left(\left(\left[x_0^k\right]_1\right)_{k=1}^{n}, [x_0]_2\right)$
- $\Pi_0 := \left(\Pi^{Agg}, \Pi^{Ind}\right) := \left([x_0]_1, ([x_0]_1, [x_0]_2)\right)$

n: circuit size

Plonk SRS Update (SU)

- Given:
  - $srs_{i-1} = \left(\left(\left[x_{i-1}^k\right]_1\right)_{k=1}^n, [x_{i-1}]_2\right)$
- Sample $\bar{x}_i \leftarrow \mathbb{Z}_p$
- Set $[x_i]_2 := \bar{x}_i \cdot [x_{i-1}]_2$;

  and $\left[x_i^k\right]_1 := \bar{x}_i^k \cdot \left[x_{i-1}^k\right]_1$ for $k = 1, 2, \dots, n$
- $srs_i \coloneqq \left(\left(\left[x_i^k\right]_1\right)_{k=1}^n, [x_i]_2\right)$
- $\Pi_i \coloneqq \left(\Pi^{Agg}, \Pi^{Ind}\right) \coloneqq \left([x_i]_1, ([\bar{x}_i]_1, [\bar{x}_i]_2)\right)$

→n multiplications in E_1, 1 multiplication in E_2

Plonk SRS Verification (Prover)

- Given:
  - $srs_i = \left(\left(\left[x_i^k\right]_1\right)_{k=1}^{n}, [x_i]_2\right)$
  - $\Pi_i = \left([x_i]_1, ([\bar{x}_i]_1, [\bar{x}_i]_2)\right)$
- For $k = 1,2, \dots, n$: check $\left[x_i^k\right]_1 \bullet [1]_2 = \left[x_i^{k-1}\right]_1 \bullet \left[x_i^1\right]_2$

→ $2n$ pairings

Plonk SRS Verification (Verifier)

n: circuit size
i: # updates

- Given:
  - $srs_i = \left(\left(\left[x_i^k\right]_1\right)_{k=1}^{n}, [x_i]_2\right)$
  - For j $= 0,1,\ldots,\mathrm{i}, \Pi_\mathrm{j} = \left(\left[x_j\right]_1, \left(\left[\bar{x}_j\right]_1, \left[\bar{x}_j\right]_2\right)\right)$
- Check that $[x_0]_1 = [\bar{x}_0]_1$
- For j $= 0,1,\ldots,$i: check $\left[\bar{x}_j\right]_1 \bullet [1]_2 = [1]_1 \bullet \left[\bar{x}_j\right]_2$
- For j $= 1,2,\ldots,$i: check $\left[x_j\right]_1 \bullet [1]_2 = \left[x_{j-1}\right]_1 \bullet \left[\bar{x}_j\right]_2$
- For j $= 1,2,\ldots,n$: check $\left[x_i^k\right]_1 \bullet [1]_2 = \left[x_i^{k-1}\right]_1 \bullet \left[x_i^1\right]_2$

→ $2n + 4i - 2$ pairings

Batched SRS Verification

- Use the following property (BellareGR98)
- If $\sum_i t_i[a_i]_1 \bullet [1]_2 = [1]_1 \bullet \sum_i t_i[a_i]_2$ for uniformly random t_i,
  then $[a_i]_1 \bullet [1]_2 = [1]_1 \bullet [a_i]_2$ for each i, with high probability
- $t_i \in \{0,1\}^{40}\ or \in \{0,1\}^{80}$ for $2^{-40}\ or\ 2^{-80}$ security

- Reduce pairing at the cost of multiplications

Plonk Batched SRS Verification (Prover)

- Given:
  - $srs_i = \left(\left(\left[x_i^k\right]_1\right)_{k=1}^{n}, [x_i]_2\right)$
  - $\Pi_i = ([x_i]_1, ([\bar{x}_i]_1, [\bar{x}_i]_2))$
- Sample $t_k \leftarrow \mathbb{Z}_p^*$ for $k = 2, \dots, n$
- Check if $\left([x_i]_1 + \sum_{k=2}^{n} t_k \cdot \left[x_i^k\right]_1\right) \bullet [1]_2 = \left([1]_1 + \sum_{k=2}^{n} t_k \left[x_i^{k-1}\right]_1\right) \bullet [x_i]_2$

→ 2 pairings instead of $2n$ at the cost of $2n - 2$ multiplications in E_1

Plonk Batched SRS Verification (Verifier)

- Given:
  - $srs_i = \left(\left([x_i^k]_1\right)_{k=1}^{n}, [x_i]_2\right)$
  - For j $= 0,1,\ldots,$ i, $\Pi_j = \left([x_j]_1, \left([\bar{x}_j]_1, [\bar{x}_j]_2\right)\right)$
- Sample $t_j, s_j \leftarrow \mathbb{Z}_p^*$ for j $= 1,\ldots,i$ and $h_k \leftarrow \mathbb{Z}_p^*$ for $k = 1,\ldots,n$
- Check that $[x_0]_1 = [\bar{x}_0]_1$
- Check if $\left([\bar{x}_0]_1 + \sum_{j=1}^{i}\left(t_j[\bar{x}_j]_1 + s_j[x_j]_1\right) + \sum_{k=1}^{n} h_k[x_i^k]_1\right) \bullet [1]_2$
  $= [1]_1 \bullet \left([\bar{x}_0]_2 + \sum_{j=1}^{i} t_j[\bar{x}_j]_2\right) + \sum_{j=1}^{i}\left(s_j[x_{j-1}]_1 \bullet [\bar{x}_j]_2\right) + \left(\sum_{k=1}^{n} h_k[x_i^{k-1}]\right) \bullet [x_i]_2$

→ $i + 3$ pairings instead of $2n + 4i - 2$ at the cost of $2n + 4i$ multiplications in E_1

Other SNARK: Sonic

If party = V:

1. Sample $\{r_{1,j}, r_{2,j}, r_{3,j}, r_{4,j} \leftarrow \mathbb{Z}_p^\star\}_{j=0}^{i}$; and $\{t_k, \hat{t}_k \leftarrow \mathbb{Z}_p^\star\}_{k=-n}^{n}$;
2. Check that $[x_0]_1 = [\bar{x}_0]_1$, $[a_0 x_0]_1 = [\bar{a}_0 \bar{x}_0]_1$, and $[a_0]_2 = [\bar{a}_0]_2$;
3. Check if $\left(\sum_{j=0}^{i} r_{1,j} \cdot [\bar{x}_j]_1\right) \bullet [1]_2 = [1]_1 \bullet \left(\sum_{j=0}^{i} r_{1,j} [\bar{x}_j]_2\right)$;
4. Check if $\left(\sum_{j=0}^{i} r_{2,j} \cdot [\bar{a}_j \bar{x}_j]_1\right) \bullet [1]_2 = [1]_1 \bullet \left(\sum_{j=0}^{i} r_{2,j} [\bar{a}_j \bar{x}_j]_2\right) =$

$^{i}_{j=0} \left(r_{2,j} \cdot [\bar{x}_j]_1 \bullet [\bar{a}_j]_2\right)$;

heck if $\left(\sum_{j=1}^{i} r_{3,j} \cdot [x_j]_1\right) \bullet [1]_2 = \sum_{j=1}^{i} \left(r_{3,j} \cdot [x_{j-1}]_1 \bullet [\bar{x}_j]_2\right)$;

heck if $\left(\sum_{j=1}^{i} r_{4,j} \cdot [a_j x_j]_1\right) \bullet [1]_2 = \sum_{j=1}^{i} \left(r_{4,j} \cdot [x_j]_1 \bullet [a_j]_2\right) =$

$^{i}_{j=1} \left(r_{4,j} \cdot [a_{j-1} x_{j-1}]_1 \bullet [\bar{a}_j \bar{x}_j]_2\right)$;

heck if $\left(\sum_{k=-n}^{n} t_k \cdot \left[x_i^k\right]_1\right) \bullet [1]_2 = [1]_1 \bullet \left(\sum_{k=-n}^{n} t_k \cdot \left[x_i^k\right]_2\right)$;

heck if $\left(\sum_{k=-n+1}^{n} t_k \cdot \left[x_i^k\right]_1\right) \bullet [1]_2 = \left(\sum_{k=-n+1}^{n} t_k \cdot \left[x_i^{k-1}\right]_1\right) \bullet [x_i]_2$;

heck if $\left(\sum_{k=-n,k\neq 0}^{n} \hat{t}_k \cdot \left[a_i x_i^k\right]_1\right) \bullet [1]_2 = [1]_1 \bullet$

$\sum_{k=-n,k\neq 0}^{n} \hat{t}_k \cdot \left[a_i x_i^k\right]_2\Big) = \left(\sum_{k=-n,k\neq 0}^{n} \hat{t}_k \cdot \left[x_i^k\right]_1\right) \bullet [a_i]_2$;

n 1 if all the checks passed, otherwise return $\perp$.

Batched SRS Verification, $(\perp/1) \leftarrow \mathsf{BSV}(\mathsf{srs}_i, (\Pi_j)_{j=0}^{i}, \mathsf{party})$:

To verify (an i-time updated) $\mathsf{srs}_i := \left(\left(\left[x_i^k\right]_1, \left[x_i^k\right]_2, \left[a_i x_i^k\right]_2\right)_{k=-n}^{n}, \left(\left[a_i x_i^k\right]_1\right)_{k=-n,k\neq 0}^{n}, [a_i]_T\right)$, and $\Pi_j := \left(\Pi_j^{\mathsf{Agg}}, \Pi_j^{\mathsf{Ind}}\right) := \left(\left([x_j]_1, [a_j x_j]_1, [a_j]_2\right), \left([\bar{x}_j]_1, [\bar{x}_j]_2, [\bar{a}_j \bar{x}_j]_1, [\bar{a}_j \bar{x}_j]_2, [\bar{a}_j]_2\right)\right)$ for $j = 0, 1, \cdots, i$:

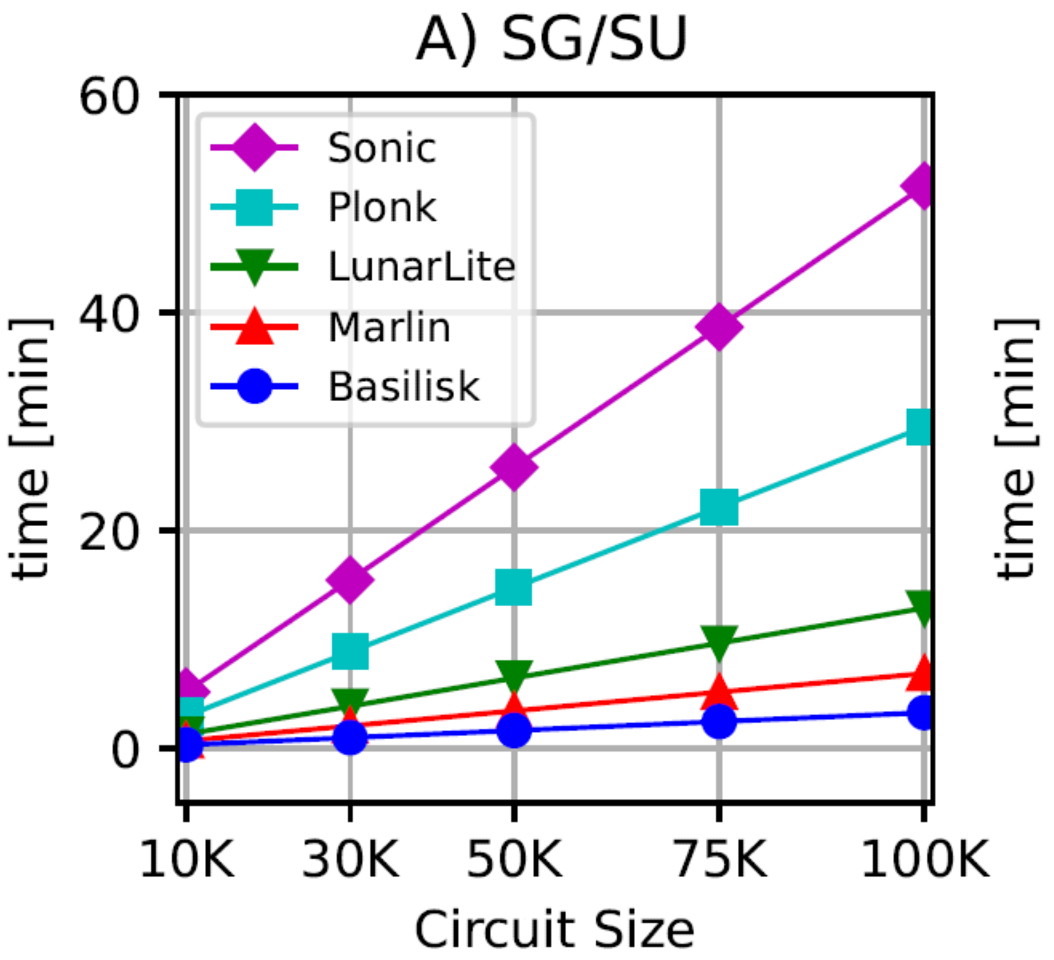
If party = P:

1. Sample $\{t_k, \hat{t}_k \leftarrow \mathbb{Z}_p^\star\}_{k=-n}^{n}$;
2. Check if $\left(\sum_{k=-n}^{n} t_k \cdot \left[x_i^k\right]_1\right) \bullet [1]_2 = [1]_1 \bullet \left(\sum_{k=-n}^{n} t_k \cdot \left[x_i^k\right]_2\right)$;
3. Check if $\left(\sum_{k=-n+1}^{n} t_k \cdot \left[x_i^k\right]_1\right) \bullet [1]_2 = \left(\sum_{k=-n+1}^{n} t_k \cdot \left[x_i^{k-1}\right]_1\right) \bullet [x_i]_2$;
4. Check if $\left(\sum_{k=-n,k\neq 0}^{n} \hat{t}_k \cdot \left[a_i x_i^k\right]_1\right) \bullet [1]_2 = [1]_1 \bullet \left(\sum_{k=-n,k\neq 0}^{n} \hat{t}_k \cdot \left[a_i x_i^k\right]_2\right) = \left(\sum_{k=-n,k\neq 0}^{n} \hat{t}_k \cdot \left[x_i^k\right]_1\right) \bullet [a_i]_2$;

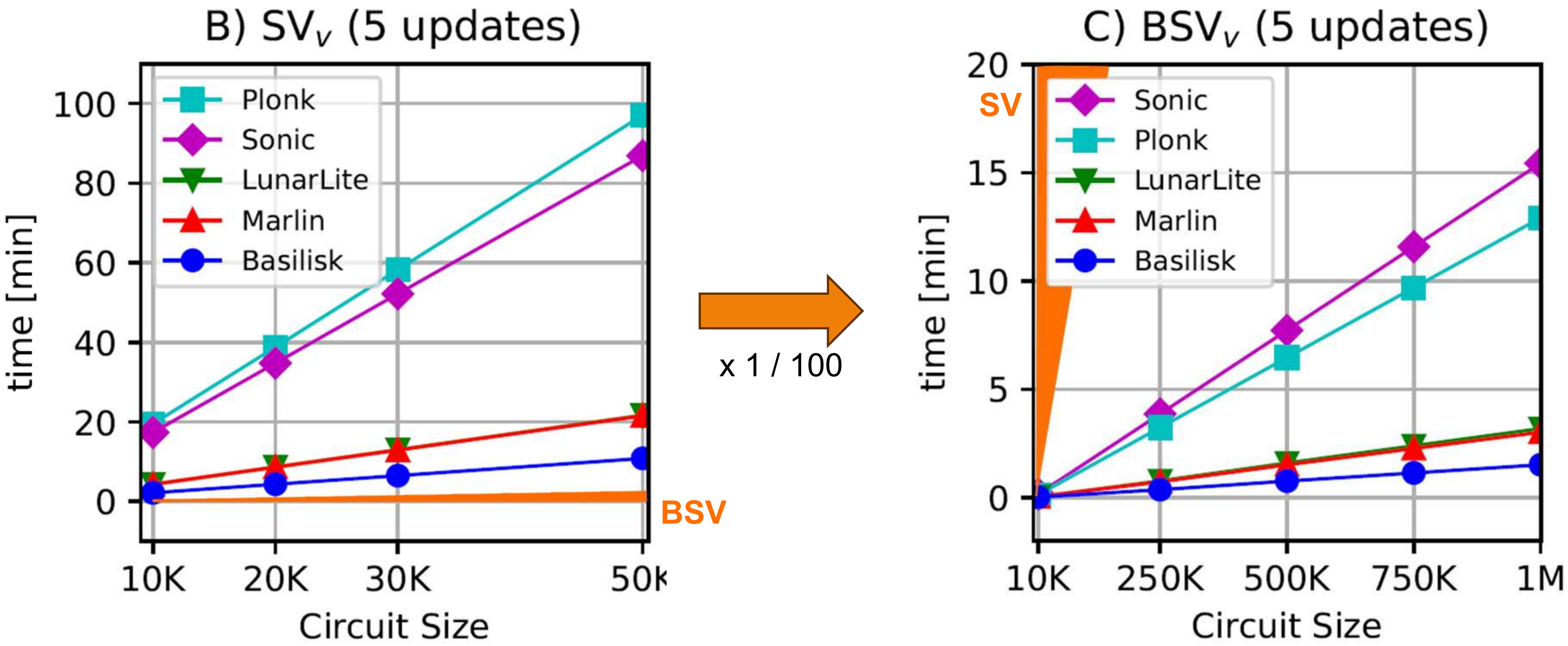
Benchmarks

- Comparison of the updatable zk-SNARKS (batched vs not batched)
- Performance of Basilisk
  - Parallelization
- Two graphs:
  - Time – # updates
  - Time – circuit size

Benchmarks

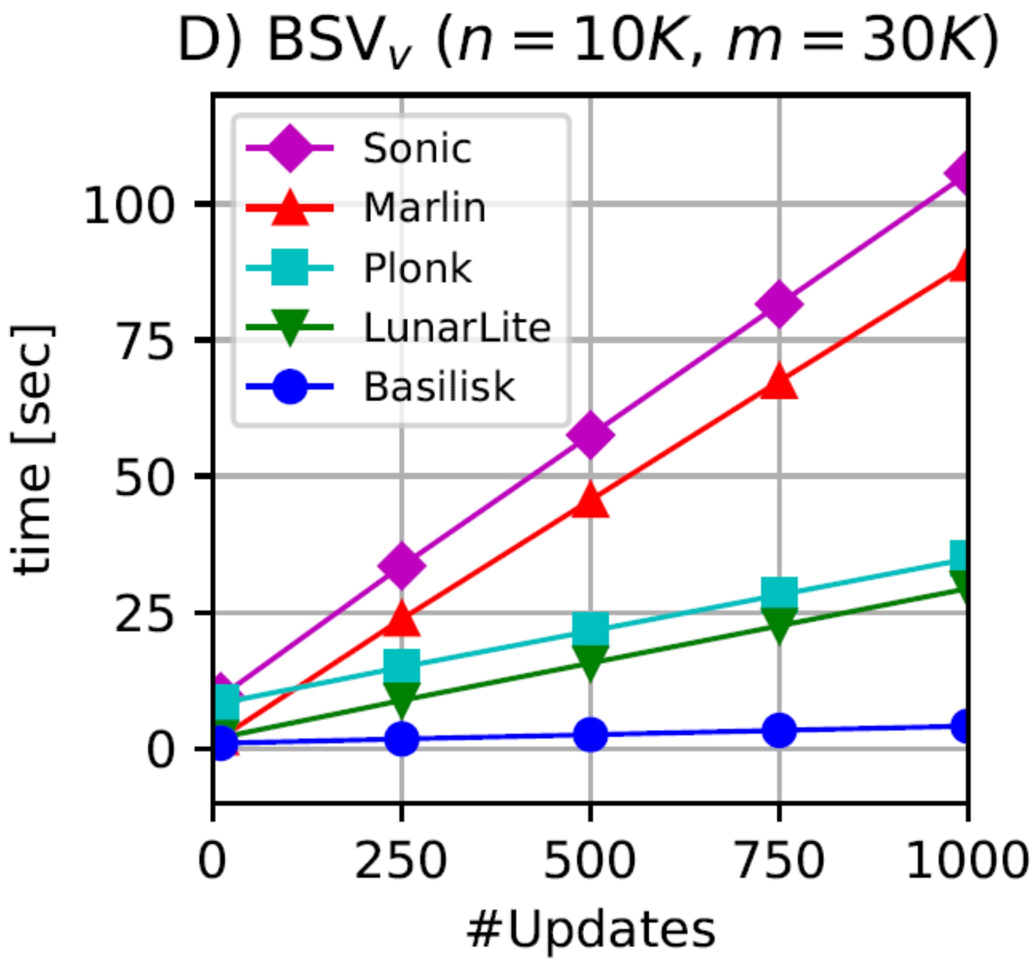

A) SG/SU
60
40
20
0
time [min]
time [min]
Sonic
Plonk
LunarLite
Marlin
Basilisk
10K
30K
50K
75K
100K
Circuit Size

Benchmarks



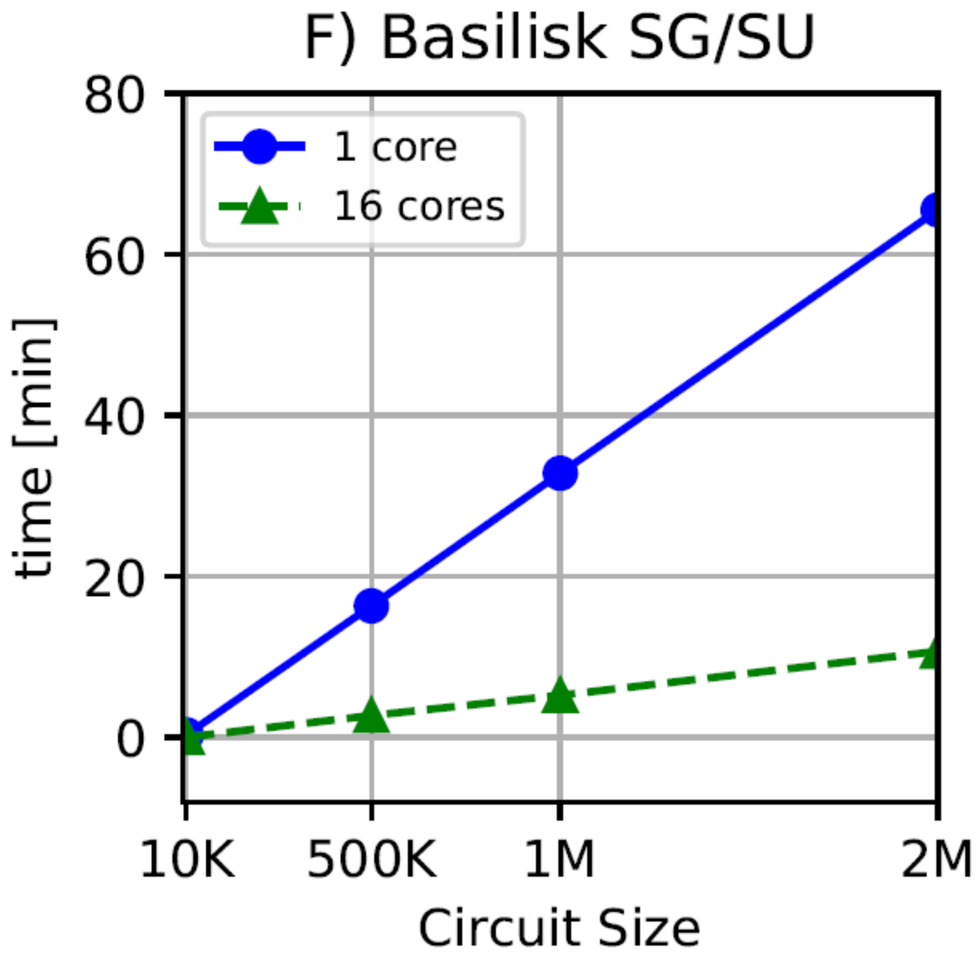
B) SV_v (5 updates)
Plonk
Sonic
LunarLite
Marlin
Basilisk
time [min]
100
80
60
40
20
0
10K
20K
30K
50K
Circuit Size
BSV
x 1 / 100
C) BSV_v (5 updates)
SV
Sonic
Plonk
LunarLite
Marlin
Basilisk
time [min]
20
15
10
5
0
10K
250K
500K
750K
1M
Circuit Size

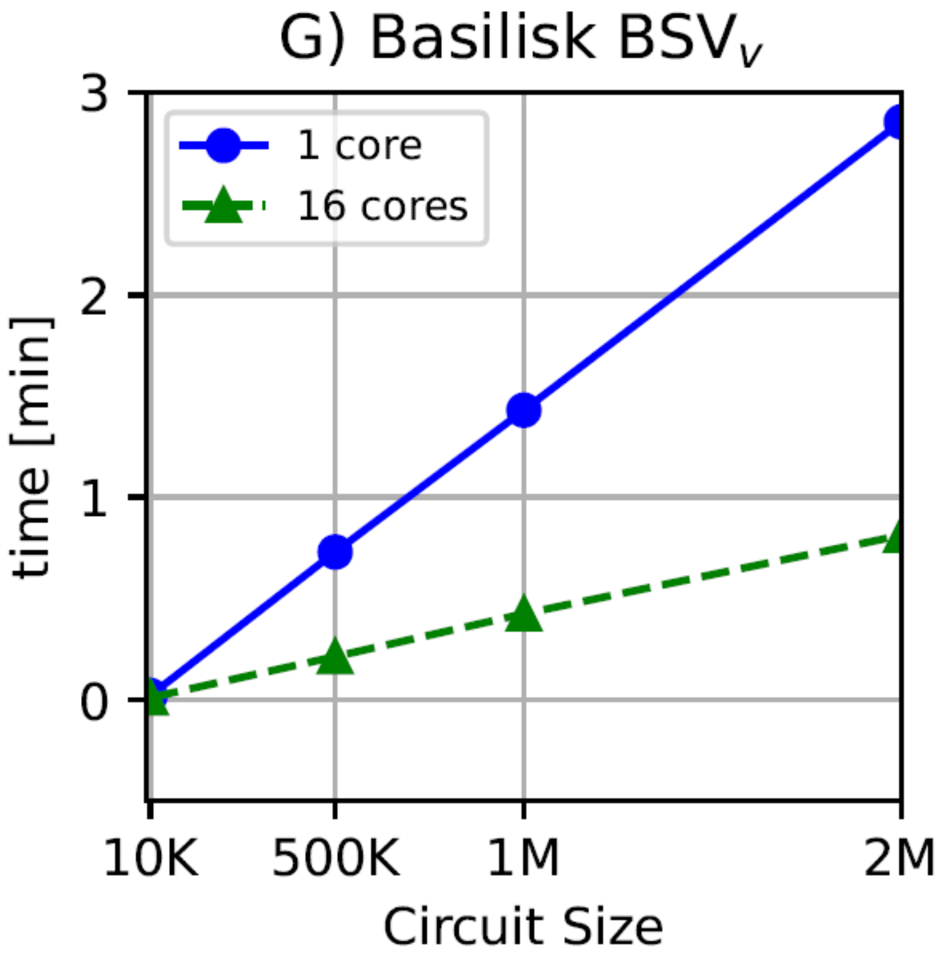
Benchmarks



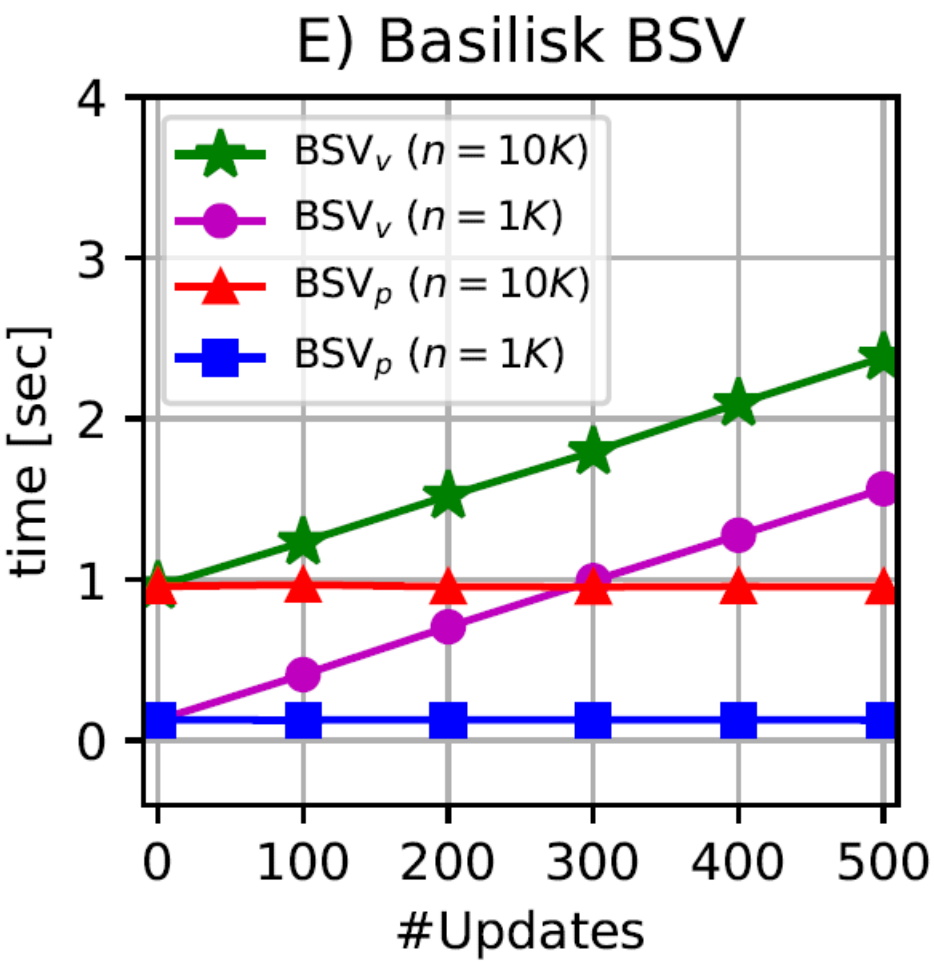
D) BSV$_v$ ($n = 10K$, $m = 30K$)
Sonic
Marlin
Plonk
LunarLite
Basilisk
100
75
50
25
0
time [sec]
0
250
500
750
1000
#Updates

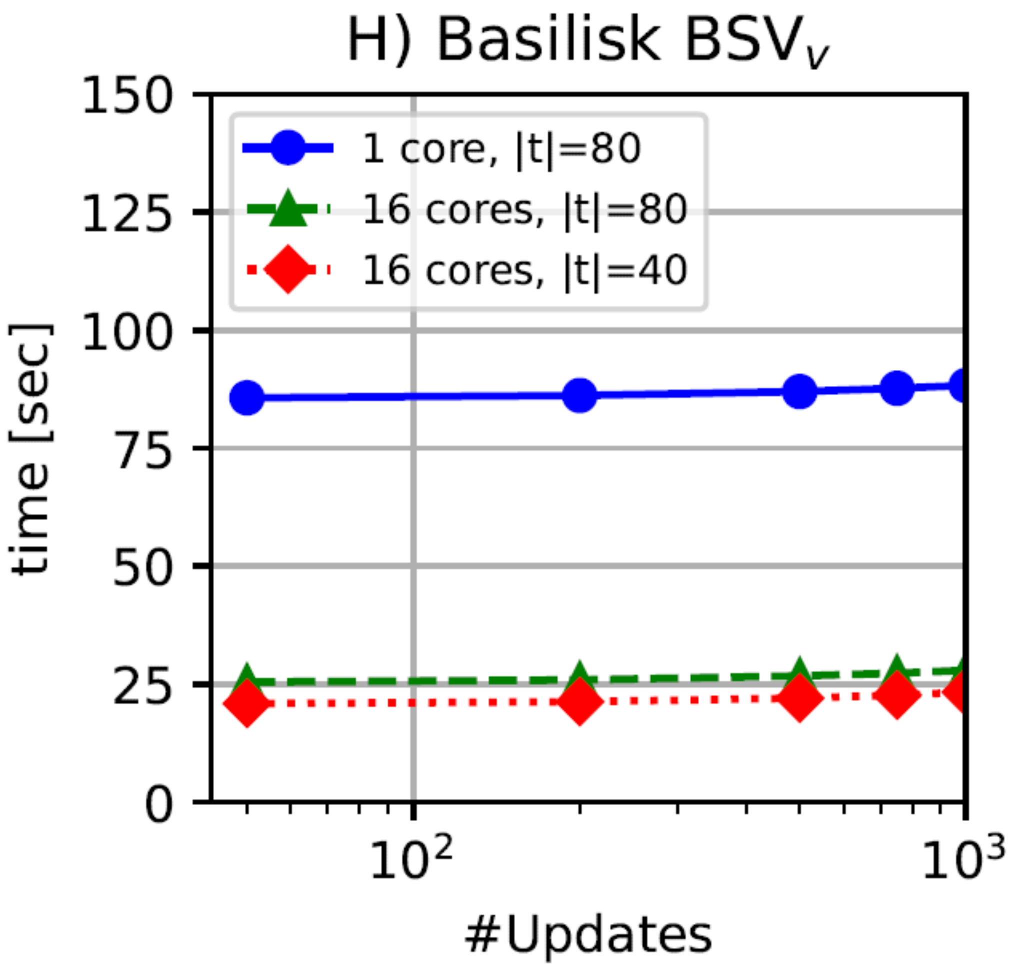
Basilisk Benchmarks





Basilisk Benchmarks





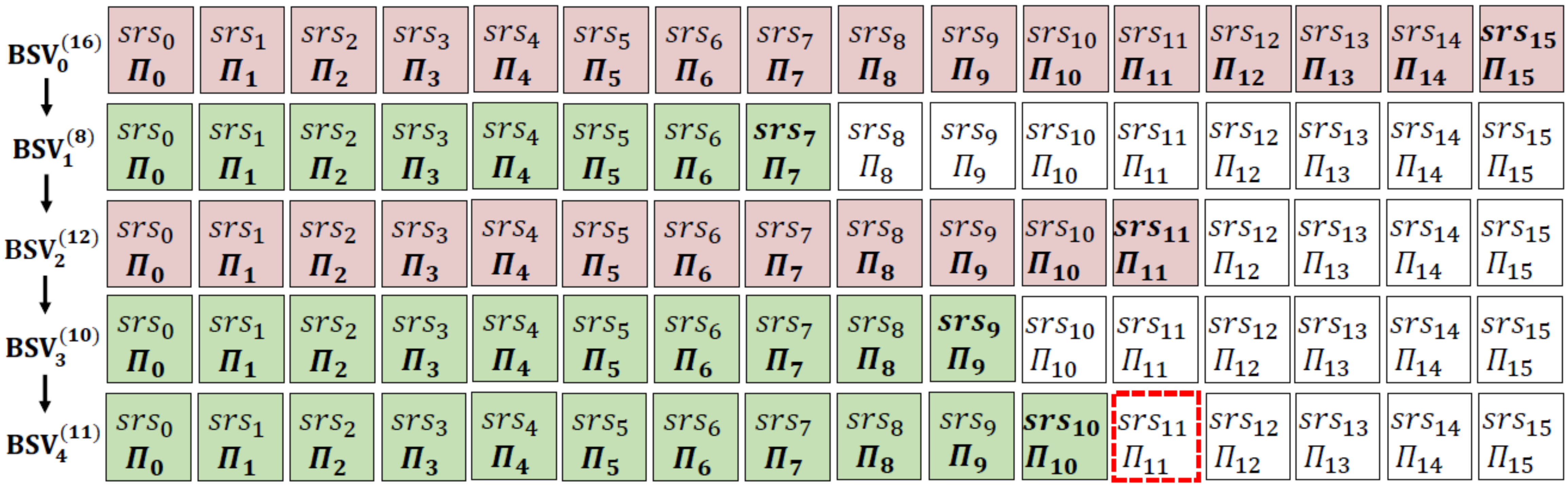
Identifiable security

- Identifying the malicious party

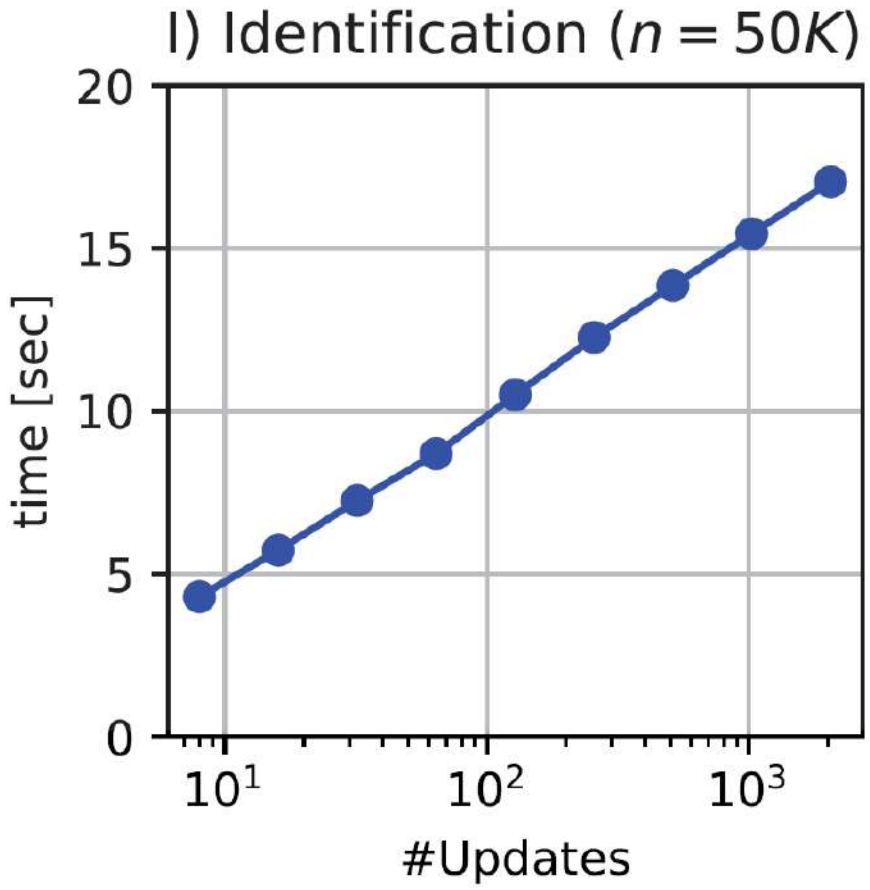
- Naïve: SV after each update
  - Scales poorly

- Binary search
  - Logarithmic in #updates
  - All transcripts need to be stored (also srs_i, not just Π_i)

Identifiable Security



Identifiable Security



Final Performance (Basilisk)

- Circuit size 2^{20} and 1000 updates
  - $SU < 6\,min$
  - $BSV_V < 30\ s$
  - Identification $< 4\,min$

- Questions?