Time-Lock Puzzles from Lattices

Shweta Agrawal[1] Giulio Malavolta[2, 3] **Tianwei Zhang** [3, 4]

[1]IIT Madras

[2]Bocconi University

[3]Max Planck Institute for Security and Privacy

[4]Ruhr University Bochum

Crypto 2024, August 17, 2024

Outline

Time-Lock Puzzles

$$\text{Puzzle(m)} \xrightarrow{\textbf{Takes time T}} m$$

- Fast puzzle generation - Time to generate Puzzle(m) is much shorter than time T (sublinear).
- Puzzle opening takes a long time - The circuit that opens Puzzle(m) has depth at least T. Parallelism shouldn't help.

Applications

Encrypt to the future!

- Sealed Bid Auctions
- Non-Malleable Commitments
- Miner extractable value prevention
- More: Blockchain front running prevention, fair contract signing, cryptocurrency payments, distributed consensus

Our Results

- (Preprocessing Model) TLP with (one-time, public-coin) Setup T, puzzle generation $\log T$.
- (Plain Model) TLP, puzzle generation time and the puzzle size $\sqrt{T}$, the first lattice-based TLP construction.
- Succinct randomized encoding (SRE) for repeated circuit computations. New Application: Sublinear Garbled RAM. Prior solution was based on iO [BGJ+16].
- Introduce the notion of range puncturable PRF.

Definition of SRE for Repeated Circuits

Definition (SRE for Repeated Circuits)

$(\tilde{C}_T, \tilde{x}) \leftarrow \mathsf{SRE.Enc}(1^\lambda, C, x, T)$: Takes time sublinear in T.

$C_T(x) = \underbrace{C(\dots C}_{T-\text{times}}(x)) \leftarrow \mathsf{SRE.Eval}(\tilde{C}_T, \tilde{x})$: Takes time T.

Security: no further information other than $C_T(x)$ is revealed about x.

TLP Circuit

The TLP circuit $C_f(b, x, m, z, i)$:
- If $i = T + 1$:
  - if $b = 0$, return m;
  - if $b = 1$, return $x \oplus z$.
- Otherwise, return $(b, f(x), m, z, i + 1)$.

We denote $C_{f,T}$ the T-fold repetition of C_f, where f is a T-folded sequential function.

TLP from SRE for Repeated Circuits

PGen(T, s): Sample $x, m, r \leftarrow \{0,1\}^\lambda$ randomly, compute $(\tilde{C}_{f,T}, \tilde{x}) \leftarrow \mathsf{SRE.Enc}(1^\lambda, C_f, (0, x, m, 0^\lambda, 1), T)$, return $Z = (\tilde{x}, r, r \cdot m \oplus s)$.

PSolve(Z): Compute $\mathsf{SRE.Eval}(\tilde{C}_{f,T}, \tilde{x}) \cdot r$ to unmask s.

Correctness:

$$C_{f,T}(0, x, m, 0, 1) = m.$$

Security of TLP

Security:

$$
\begin{aligned}
(\tilde{C}_{f,T}, \tilde{x}) &= \mathrm{Encode}(C_f, (0, x, m, 0, 1)) \\
&\equiv \mathrm{Encode}(C_f, (0, x, m \oplus f_T(x), 0, 1)) \\
&\approx \mathrm{Encode}(C_f, (0, x, m \oplus f_T(x), m, 1)) \\
&\approx \mathrm{Encode}(C_f, (1, x, m \oplus f_T(x), m, 1)) \\
&\approx \mathrm{Encode}(C_f, (1, x, 0, m, 1)) \text{ (encoding } f_T(x) \oplus m\text{)}
\end{aligned}
$$

Therefore any adversary that is able to output m in time less than T will also compute $f_T(x)$, thus violating the sequentiality of f.

Apply the depth-preserving Goldreich-Levin theorem in the reduction.

Depth-Independent Reusable Garbled Circuit

Circular small-secret LWE

$\Longrightarrow$ rGC [HLL23] + LFE [QWW18]

$\Longrightarrow$ Depth-Independent Reusable GC:

$(\tilde{C}, \mathsf{pk}) \leftarrow \mathsf{rGC.Garble}\left(1^\lambda, C\right)$, $|\mathsf{pk}| = \mathsf{poly}(\lambda)$, takes time $\mathsf{poly}(\lambda) \cdot |C|$

$\tilde{x} \leftarrow \mathsf{rGC.Enc}(\mathsf{pk}, x)$, takes time $\mathsf{poly}(\lambda) \cdot |x| \cdot |y|$

$C(x) \leftarrow \mathsf{rGC.Eval}\left(\tilde{C}, C, \tilde{x}\right)$, takes time $\mathsf{poly}(\lambda) \cdot |C|$

Security: $\mathcal{A}(\tilde{C}, \mathsf{pk}, \tilde{x}) \approx \mathcal{A}(\tilde{C}, \mathsf{pk}, \mathbf{Sim}(1^\lambda, C, \mathsf{pk}, C(x)))$

TLP with Setup

PSetup($1^\lambda, T$):
Compute $(\tilde{C}_{f,T}, \mathsf{pk}) \leftarrow \mathsf{rGC.Garble}\left(1^\lambda, C_{f,T}\right)$.

PGen(pp, s):
Sample $x, m, r \leftarrow \{0,1\}^\lambda$ randomly, compute
$\tilde{x} \leftarrow \mathsf{rGC.Enc}(\mathsf{pk}, (0, x, m, 0^\lambda, 1))$, return $Z = (\tilde{x}, r, r \cdot m \oplus s)$.

PSolve(Z):
Compute $\mathsf{rGC.Eval}\left(\tilde{C}_{f,T}, C_{f,T}, \tilde{x}\right) \cdot r$ to unmask s.

Attempt to construct SRE for Repeated Circuits

Idea: Reuse the preprocessing to *amortize the work.*

Circuit $F_{\sqrt{T}}(x, i)$:
If $i = T + 1$ return x;
else compute $y = f_{\sqrt{T}}(x)$, output an *encoding* of $(y, i + \sqrt{T})$.

"SRE.Encode":
Compute $(\tilde{F}_{\sqrt{T}}, \mathrm{pk}) \leftarrow \mathrm{rGC.Garble}(1^\lambda, F_{\sqrt{T}})$,
output $\mathrm{rGC.Enc}(\mathrm{pk}, (\mathrm{x}, 1))$.

"SRE.Decode":
encoding of $(x, 1) \rightarrow$ encoding of $(f_{\sqrt{T}}(x), \sqrt{T} + 1) \rightarrow \cdots \rightarrow f_T(x)$

Problem with the attempt

Problem

However, the size of an encoding in [HLL23] depends on the output size of the circuit, which means that it grows *exponentially* with the number of repetitions!

To fix this, we use split-FHE [BDGM23]: when evaluating $\mathsf{Enc}(m) \to \mathsf{Enc}(g(m))$, one can compute a *small hint* $\mathsf{h}_{g,m}$ ($|\mathsf{h}_{g,m}|$ is independent of $|g(m)|$) that allows one to decrypt the evaluated ciphertext.

Construction of SRE for Repeated Circuits

Modify $F_{\sqrt{T}}$ to output the hint of the split-FHE computation:

If $i = \sqrt{T} + 1$: Return x.

Otherwise, compute $c \leftarrow \text{split-FHE.Eval}(\Gamma_{\mathsf{i},\bar{\mathsf{pk}}}(\cdot), \mathsf{c_i})$.

Return a masked small hint h_i of c.

The circuit $\Gamma_{i,\bar{\mathsf{pk}}}(x, K\{i+1\})$:

- Compute $y = f_{\sqrt{T}}(x)$.
- Return FHE ciphertext c_{i+1} of $(y, K\{i+2\})$ and rGC encoding e_{i+1} of $(y, i+1, \bar{\mathsf{pk}}, c_{i+1})$.

Construction of SRE for Repeated Circuits Continued

$\mathsf{SRE.Enc}(1^\lambda, \mathsf{f}, \mathsf{x}, \mathsf{T})$

Output

the garbled circuit: $(\tilde{F}_{\sqrt{T}}, \mathrm{pk}) \leftarrow \mathrm{rGC.Garble}\left(1^\lambda, F_{\sqrt{T}}\right)$,
the garbled input: a FHE ciphertext c_1 of x and a rGC encoding e_1 of $(\mathrm{pk}_1, x, 1, \mathrm{pk}, c_1)$.

$\mathsf{SRE.Dec}(1^\lambda, \mathsf{f}, \mathsf{x})$

- For $i = 1, \ldots, \sqrt{T}$:
  Compute $c \leftarrow \mathrm{FHE.Eval}(\mathrm{pk}_i, \Gamma_{i,\mathrm{pk}}, c_i)$.
  Decode $h_i \leftarrow \mathrm{rGC.Eval}(\tilde{F}_{\sqrt{T}}, F_{\sqrt{T}}, e_i)$.
  Get (c_{i+1}, e_{i+1}) by decrypting h_i and c.
- Output $\mathrm{rGC.Eval}\left(\tilde{F}_{\sqrt{T}}, F_{\sqrt{T}}, e_{\sqrt{T}+1}\right)$.

Conclusion

- TLP with Setup T, puzzle generation $\log T$. TLP: puzzle generation $\sqrt{T}$.
- Introduce range puncturable PRF and SRE along the way.
- Heuristic Fully Efficient SRE, hence TLP with $\log T$ puzzle generation time.

Open Problems

- lattice-based fully efficient SRE, hence TLP with $\log T$ puzzle generation time.
- lattice-based homomorphic TLPs.
- lattice-based batch-solving TLPs.

Questions?

Thank you for your attention!

*Q*uestions?