

Post-Quantum Time-Lock Puzzle from Isogenies

Shweta Agrawal^{*} Andrea Basso^{**} Sikhar Patranabis^{***}

Abstract. We construct the first (conjectured) post-quantum Time Lock Puzzle (TLP) from isogenies. Our construction avoids setup/preprocessing and achieves non-trivial efficiency, where puzzle generation runs in time $\sqrt{T}$. We prove security in the quantum random oracle model (QROM) under a new conjecture about the sequentiality of isogeny pushforward computations, for which we provide a thorough justification. The only other (conjectured) post-quantum TLP constructions rely on heavy cryptographic machinery from lattice-based assumptions [3,10].

We also provide, to the best of our knowledge, the first proof-of-concept implementation of a candidate post-quantum TLP. On an Apple M3 CPU, for a one-hour target delay, total puzzle generation, including setup, takes 2.9 minutes serially and 100.5 seconds with four generation workers. Our experiments demonstrate the practicality of the complete design and show that the quadratic gap between puzzle generation and solving in theory also translates into a meaningful concrete separation.

As a stepping stone towards our main result, we provide a new TLP from isogenies in the preprocessing model from a combination of the sequentiality of isogeny walks in the F_{p^2} graph and the hardness of computational Diffie-Hellman (CDH) over cyclic prime-order groups. This improves the assumptions underlying the only known TLP from isogenies in the preprocessing model [14], which required pairing based assumptions instead of just discrete log. We further adapt this construction to achieve non-trivial efficiency of $\sqrt{T}$ in puzzle generation without setup/preprocessing.

^{*} IIT Madras, India, shweta@cse.iitm.ac.in

^{**} IBM Research Europe, Zürich, Switzerland, andrea.basso@ibm.com

^{***} IBM Research, Bengaluru, India, sikhar.patranabis@ibm.com

Table of Contents

Post-Quantum Time-Lock Puzzle from Isogenies	1
<i>Shweta Agrawal Andrea Basso Sikhar Patranabis</i>	
1 Introduction.....	3
1.1 Prior Work	3
1.2 Our Contribution	4
2 Technical Overview.....	5
2.1 Compilation to $\sqrt{T}$ -TLP without Setup	5
2.2 Classically Secure TLP without Setup.....	7
2.3 Post-Quantum TLP from Isogenies	9
2.4 Proof-of-Concept Implementation	15
3 Preliminaries	15
3.1 Plain Time-Lock Puzzle (TLP) without Setup/Preprocessing ...	15
3.2 TLP with Setup/Preprocessing.....	16
3.3 Isogeny Graphs over $\mathbb{F}_{p^2}$	17
4 TLP without Setup from Symmetric TLP with Setup	17
4.1 Symmetric TLP with Setup.....	17
4.2 Symmetric TLP with Setup to Plain TLP without Setup	19
5 Post-Quantum Symmetric TLP with Setup.....	24
5.1 Construction of Isogeny-based Symmetric TLP with Setup	25
5.2 Finding a Shortcut	26
5.3 Parameters and Efficiency	28
5.4 Sequentiality Assumption and Security	29
6 Proof-of-Concept Implementation and Benchmarks	33
A Post-Quantum Security of the Compiler in the QROM.....	39
A.1 Technical Overview of the QROM Extension	39
A.2 The Quantum Random Oracle Model	42
A.3 Post-Quantum Security Definitions for TLP Variants	45
A.4 The QROM Analogue of Theorem 1	47
A.5 Discussion	52
A.6 Dynamic Finite-Family Simulation of the QROM	53
A.7 Proof of Theorem 5 (Post-Quantum Security in the QROM)	54
B Isogeny-based Classically Secure TLP with Setup	56
B.1 The Sequentiality Assumption	56
B.2 Construction of Classically Secure TLP with Setup	59
B.3 Efficiency Analysis	59
B.4 Correctness Analysis.....	60
B.5 Security Analysis.....	61

1 Introduction

Timed cryptography seeks to design primitives so that they are only secure for a predetermined amount of time, denoted by T . Among the first questions posed by this fascinating field is - can we encrypt messages “to the future”, so that the secret key is, somehow, *time*? A bit more formally, can we construct a ciphertext which cannot be decrypted by anyone until some fixed time (say 50 years) has elapsed, and can be decrypted by everyone after that? The seminal work of Rivest, Shamir and Wagner (RSW) [30] answered this question in the affirmative via the beautiful new notion of *time-lock puzzle*.

A time-lock puzzle (TLP) comprises a puzzle generation algorithm and a puzzle evaluation algorithm defined as follows: (i) the puzzle generation algorithm takes as input a time parameter T together with the security parameter 1^λ and a solution s and outputs an encoding or puzzle Z , (ii) the puzzle evaluation takes a puzzle Z and outputs a solution s . For efficiency, it is required that the time needed to generate a puzzle must be sublinear, ideally logarithmic in T while for security (or sequentiality), it is required that the time needed to solve a puzzle must be greater than T . Here, it is crucial that sequentiality does not succumb even to massively parallel algorithms. TLPs have been intensely studied owing to several applications including sealed-bid auctions, fair contract signing, non-malleable commitments, cryptocurrency payment systems and distributed consensus algorithms and several others - please see [3,27] for a detailed discussion.

1.1 Prior Work

The Repeated Squaring Assumption. The seminal work of [30] proposed a new assumption called the “repeated squaring assumption”. At a high level, this assumption postulates that computing $x^{2^T} \bmod N$, where $N = pq$ is an RSA modulus, takes at least T time steps. Note that it is easy, given x , to compute the desired output in T steps, simply by repeated squaring - the conjecture, as the name suggests, is that this is the fastest algorithm for the task, so long as the order of the RSA group remains unknown. The authors of [30] used this conjecture to construct the first candidate time-lock puzzle (TLP).

Lattice Assumptions. The very recent work by Agrawal, Malavolta and Zhang [3] presented the first lattice-based solution, in the *preprocessing* model under the circular Learning With Errors assumption. In this model, there is a one-time setup algorithm whose runtime can depend on T . This setup produces some small public parameters (of size independent of T) that anyone can use to encrypt a message for time T . In the plain model, they obtain $\sqrt{T}$ encoding complexity, which was improved by the subsequent work by Bitansky and Garg [10] to T^ϵ for arbitrarily small constant $\epsilon < 1$. This construction has the advantage of being plausibly post-quantum secure.

Indistinguishability Obfuscation. Bitansky et al. [11] gave constructions of TLPs of varying strengths using various flavors of randomized encodings. Their fully

efficient constructions are based on indistinguishability obfuscation (iO) and one-way functions.

Isogeny-Based Assumptions. Burdges and De Feo [14] recently put forward the notion of “Delay Encryption” (DE) which can be thought of as “Time-Lock Identity Based Encryption”. They also provided the first instantiation based on isogenies of supersingular curves and pairings. Delay encryption is defined with a setup which is slow in that it is allowed to run in time $O(T)$, hence it does not imply the standard notion of TLP which requires puzzle generation to be sublinear in T . While it does imply TLP in the weaker *pre-processing* model described above, the resulting construction is not post-quantum secure due to its implicit reliance on pairings and the hardness of discrete-log. Thus, a major question left open by prior work is:

Can we build post-quantum TLP from isogenies?

1.2 Our Contribution

In this work, we answer the above question in the affirmative. We provide the first candidate construction for post-quantum secure TLP (without setup/preprocessing) from isogeny-based assumptions. Our construction achieves non-trivial efficiency, where puzzle generation runs in time $\sqrt{T} \text{poly}(\log T, \lambda)$. We prove security in the quantum random-oracle model (QROM) [12] under a new conjecture about the sequentiality of pushforward isogeny computations, for which we provide a rigorous justification. We are optimistic that our assumption will pave the way for new post-quantum constructions of timed cryptography from isogenies.

We complement our theoretical results with, to the best of our knowledge, the first proof-of-concept implementation of a candidate post-quantum TLP. Our SageMath prototype (available anonymously at: <https://anonymous.4open.science/r/pqTLP>) uses: (i) concrete parameters motivated by the SQIsign level-I analysis [1,35]; (ii) efficient two-dimensional representations; and (iii) high-degree isogeny-walk chunks. For each target delay, we optimize the construction parameters and exploit parallelism in puzzle generation, while setup and solving remain sequential. On an Apple M3 CPU, generating a puzzle with a one-hour target delay takes 2.9 minutes serially and 100.5 seconds with four generation workers, including setup. These results demonstrate a meaningful concrete generation-solving gap, while exposing a time-size tradeoff from parallelization. See Section 6 for our detailed experimental evaluation. We stress that this is a proof of concept; a production-level implementation will require further optimization, parameter analysis, and evaluation on the intended solving hardware.

Along the way, we simplify the isogeny-based TLP in the preprocessing model by [14]. In detail, while [14] relies on a combination of the sequentiality of isogeny walks in the F_{p^2} isogeny graph and pairing-based assumptions, we present a simplified version of their TLP that relies on sequentiality of isogeny walks and the hardness of computational Diffie-Hellman (CDH) over cyclic prime-order groups. This can be seen as the timed analog of the corresponding

weakening of assumptions from the Boneh-Franklin IBE to the ElGamal PKE. More importantly, we achieve a TLP without setup/preprocessing that has non-trivial efficiency, where puzzle generation runs in time $\sqrt{T} \text{poly}(\log T, \lambda)$, from the same assumptions. No (even) classically secure TLP construction in the plain model was known before from isogenies, to the best of our knowledge.

2 Technical Overview

In this section, we outline the core technical ideas developed in this work.

2.1 Compilation to $\sqrt{T}$ -TLP without Setup

We begin with an overview of our first contribution, namely, a compiler that compiles any generic TLP in the *preprocessing model*, with a long (i.e., $O(T)$) setup and efficient (i.e., $\text{poly}(\lambda)$, independent of T) puzzle generation, into a TLP without preprocessing/setup with $O(\sqrt{T})$ puzzle generation efficiency. Our transformation relies only on random oracles.

The starting point of our work is the compiler by Bitansky and Garg [10] which shows how to convert a TLP with setup to a non-trivially efficient TLP without setup/preprocessing using only garbled circuits together with a “laconic function evaluation” (LFE) scheme that satisfies a natural efficiency preservation property and can be instantiated from lattice-based assumptions [25]. Our approach relies crucially on the observation that, while their compiler relies heavily on LFE for a recursive composition step that is crucial to obtaining efficiency T^ϵ , unrolling the compiler to perform only a *single* step that would yield a TLP without setup with $\sqrt{T}$ efficiency can allow us to use a base scheme without any advanced structural properties. This is important because the advanced machinery of LFE appears far outside the reach of isogenies.

The Compiler. We proceed to outline the compiler from TLP with setup to $\sqrt{T}$ -efficient TLP without setup/preprocessing. At a high level, puzzle solving, which must take time $O(T)$, is broken into a chain of $\sqrt{T}$ chunks of size $\sqrt{T}$ each, such that each chunk must be “unmasked and processed” before the solver can proceed to the next chunk. Sequentiality is enforced using the random oracle and smaller TLPs, so that solving the first TLP yields an intermediate key that, when queried using the random oracle, allows to “unmask” the next TLP. Solving this TLP in turn reveals the next intermediate key to be queried using the random oracle, and so on. Solving the final TLP in this chain yields the puzzle solution. Thus, each chunk entails solving a puzzle in time $O(\sqrt{T})$ and then querying the random oracle on the puzzle solution in time independent of T .

A bit more concretely, let $(\text{PSetup}, \text{PGen}, \text{PSolve})$ be a TLP with setup that satisfies correctness, efficiency and (reusable-)security with respect to a sequentiality parameter T_0 . Also, let H be a suitable hash function, modeled as a random oracle. Our compiler generates a TLP without setup as outlined below (see [Section 4.2](#) for the detailed description).

Puzzle Generation. Given a solution $s \in \{0, 1\}^\lambda$, proceed as follows:

1. Run $\text{pp} \leftarrow \text{PSetup}(1^\lambda, T_0)$ internally and set $s_{T_1} = s$.
2. Sample independent “masking” keys $k_1, \dots, k_{T_1-1} \leftarrow \{0, 1\}^\lambda$ and set $s_i = k_i$ for $i < T_1$.
3. Generate $Z_i \leftarrow \text{PGen}(\text{pp}, s_i)$ for every $i \in [T_1]$.
4. For $i = 2, \dots, T_1$, compute $\text{ct}_i = Z_i \oplus H(k_{i-1})$, using a canonical, fixed-length encoding of each puzzle description.
5. Output $Z^* = (Z_1, \text{ct}_2, \dots, \text{ct}_{T_1})$.

Puzzle Solver. The compiled puzzle solver takes as input a puzzle Z^* , and proceeds as follows for each $i = 1, \dots, T_1 - 1$ in sequence:

1. Compute $k_i = \text{PSolve}(Z_i)$.
2. Recover $Z_{i+1} = \text{ct}_{i+1} \oplus H(k_i)$.

Finally, it outputs $\text{PSolve}(Z_{T_1}) = s$. Correctness follows immediately from the correctness of the underlying TLP with setup.

Efficiency. For the underlying TLP with preprocessing, setup takes $O(T_0)$ time, while the corresponding puzzle-generation time does not depend on T_0 . The compiler uses T_1 underlying puzzles. Thus, for the compiled TLP, puzzle generation takes $(T_0 + T_1) \text{poly}(\log T, \lambda)$. On the other hand, the compiled TLP solver solves the T_1 puzzles in order, where each solution takes $O(T_0)$ time. Hence, the overall puzzle solving time for the compiled TLP is $T_0 T_1 \text{poly}(\lambda)$. Let $T = T_0 T_1$. The balanced choice of $T_0 = T_1 = \sqrt{T}$ gives $\sqrt{T} \text{poly}(\log T, \lambda)$ puzzle generation and $T \text{poly}(\lambda)$ puzzle solving, as desired.

Security in the ROM. We first present an informal sketch of the security of the compiled TLP in the ROM against classical adversaries. The proof relies on the following observations: (i) before query to the random oracle H on the intermediate key k_i , the value $H(k_i)$ is fresh and uniformly random from the point of view of the adversary in the compiled TLP security experiment; (thus, ct_{i+1} is a perfect one-time pad for Z_{i+1}); (ii) before Z_{i+1} is revealed to the adversary, the intermediate key k_{i+1} is uniform and independent of its view; and (iii) the probability that two intermediate keys collide is negligible by a simple birthday bound argument.

Based on these observations, we first argue the following: (i) assuming no two intermediate keys among $k_1, \dots, k_{T_1-1}$ are equal, an adversary must query $H(k_i)$ before the puzzle Z_{i+1} is revealed to it, and (ii) the adversary must query the random oracle on $k_1, \dots, k_{T_1-1}$ in sequence in order to learn any information about Z_{T_1} , and hence, about the challenge solution in the compiled TLP security experiment. Conditioned on the intersection of events (i) and (ii), we then invoke a pigeonhole style-argument that the adversary must solve, with non-negligible probability, some underlying puzzle in parallel time $o(T_0)$.

More precisely, the reduction chooses a random position i^* in the chain and embeds a single reusable-security challenge for the underlying TLP with setup at

that position. It simulates the preceding portion of the compiled puzzle using its reusable puzzle-generation oracle, and uses the adversary’s progress through the chain to distinguish which intermediate key was hidden in the challenge puzzle. Since i^* is chosen at random, the averaging argument above ensures a non-negligible probability of selecting a position that the adversary crosses sufficiently quickly, yielding an $o(T_0)$ -depth attack on the reusable security of the underlying TLP with setup. See [Theorem 1 \(Section 4.2\)](#) and its proof in the ROM ([Section 4.2](#)) for the detailed analysis.

Security in the QROM. Since our final goal is to achieve a post-quantum TLP without setup via this compiler, we also present a proof of QROM security for the compiler. Extending the above proof argument in the classical ROM to the QROM requires some care. In particular, the classical ROM proof reasons about the first time the adversary queries $H(k_i)$, but such a query is not directly observable when the adversary accesses H in superposition.

To address this issue, we divide the quantum adversary’s depth into a fixed sequence of short blocks, where each block corresponds to an intermediate key k_i . We then consider a sequence of games in which the hash value associated with each intermediate key k_i is activated only after a prescribed block. Each pair of adjacent games differ at only one oracle point over a fixed prefix of the computation. We then invoke standard QROM switching bounds to relate the distinguishing gap between these games to the adversary’s total query weight on the corresponding intermediate key. In particular, we aggregate these gaps with a single Cauchy-Schwarz argument, which allows us to avoid accumulating a square-root loss at every link of the chain.

A non-negligible distinguishing advantage therefore implies that, for some link, the adversary places noticeable query weight on its intermediate key within one of these short blocks. We turn this observation into an attack on the underlying TLP with setup by embedding one reusable-security challenge at a randomly chosen link and measuring a randomly chosen oracle query within the corresponding block. Noticeable query weight then gives a noticeable probability of extracting the hidden intermediate key, while the reduction’s depth is only roughly the adversary’s total depth divided by the number of links, and hence is $o(T_0)$. This contradicts post-quantum reusable security of the underlying TLP. See [Appendix A](#) for the detailed argument.

2.2 Classically Secure TLP without Setup

As a first step towards our post-quantum isogeny-based TLP, we present a classically secure isogeny-based TLP with setup. It relies on a deterministic, multi-use variant of the isogeny-shortcut assumption from prior works on isogeny-based verifiable delay functions [20,17], which in turn requires assuming the classical hardness of CDH over supersingular elliptic curves. The compiler described above converts this TLP with setup into a classically secure TLP without setup that has $\sqrt{T}$ efficiency. To our knowledge, this is the first (even classically secure) TLP *without setup/preprocessing* based on isogeny assumptions.

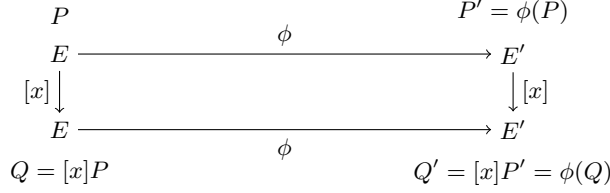


Fig. 1. Warmup: Construction of classically secure TLP with setup from the deterministic multi-use isogeny-shortcut assumption.

The Deterministic Multi-Use Isogeny-Shortcut Assumption. Let λ be the security parameter, and let T be the sequentiality parameter. Let $p = O(2^{\text{poly}(\lambda)})$ be a prime such that $(p+1)$ contains a large prime factor $N = \Theta(2^{2\lambda})$. Let $\ell = O(1)$. Let $E/\mathbb{F}_{p^2}$ be a supersingular curve for which CDH is hard over $E[N]$. Sample a uniform public seed $\text{seed}_\phi \leftarrow \{0, 1\}^\lambda$ and let $\phi = \text{DetIsog}(E, \ell, T; \text{seed}_\phi) : E \rightarrow E'$ be the resulting cyclic ℓ^T -isogeny. Let $\text{pp} = (p, N, \ell, T, E, E', \text{seed}_\phi, P, P')$ where $P \in E[N]$ is a random point, $P' = \phi(P)$. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be a two-phase, probabilistic polynomial-time (PPT) adversary such that $\mathcal{A}_1$ is allowed to preprocess pp without any depth-restriction, while $\mathcal{A}_2$ is depth- $o(T)$ -bounded and must output $\phi(Q)$ given a fresh, uniform $Q = [x]P$ and some state from $\mathcal{A}_1$. The deterministic, multi-use isogeny-shortcut assumption states that $\mathcal{A}_2$ successfully outputs $\phi(Q)$ with only negligible probability. See [Assumption 2](#) for the formal description.

The above assumption has two main distinctions with the original isogeny-shortcut assumption from [20, 17]. First, the isogeny ϕ is deterministically generated from a uniform public seed (see [Algorithm 1](#) for the detailed procedure). Additionally, the assumption has a multi-use flavor where preprocessing phase of the adversary can evaluate ϕ on polynomially many arbitrary points of its choice, while the depth-bounded online phase must evaluate ϕ on a fresh, uniformly sampled challenge point. This two-stage, multi-use formulation of the assumption is necessary for arguing reusable security of our TLP with setup. For a more detailed discussion, see [Appendix B.1](#).

Our Classically Secure TLP with Setup. We now outline our classically secure TLP in the preprocessing model (see [Fig. 1](#) for a succinct representation). Let H denote a hash function, modeled as a random oracle.

- Setup samples $\text{seed}_\phi \leftarrow \{0, 1\}^\lambda$, computes $\phi = \text{DetIsog}(E, \ell, T; \text{seed}_\phi)$ using [Algorithm 1](#), computes $P' = \phi(P)$, and publishes public parameters

$$\text{pp} = (p, N, \ell, T, E, E', \text{seed}_\phi, P, P').$$

This takes $T \text{ poly}(\lambda)$ time. Note that pp does not contain the full T -step description of ϕ , which can be deterministically re-generated given $(E, \ell, T, \text{seed}_\phi)$.

- Given pp and a λ -bit solution s , the puzzle generator samples $x \leftarrow [0, N-1]$ and publishes (pp, Q, e) , where $Q = [x]P$ and $e = s \oplus H([x]P')$. This takes $O(\log N) = O(\lambda)$ time, independently of T .

- The puzzle solver reconstructs $\phi = \text{DetIsog}(E, \ell, T; \text{seed}_\phi)$ using [Algorithm 1](#) and computes $Q' = \phi(Q) = [x]P'$. It recovers $s = e \oplus H(Q')$.

By the deterministic multi-use isogeny-shortcut assumption, the computation of Q' during solving takes T sequential steps. Thus puzzle solving takes $T \text{poly}(\lambda)$ time. See [Appendix B](#) for a more detailed description and analysis.

The above TLP implicitly relies on the hardness of computing discrete logarithms and is hence not post-quantum secure: in particular, a quantum adversary could also solve the puzzle by recovering the scalar x given $(P, Q = [x]P)$ and then computing $Q' = [x]P'$. Next, we discuss approaches to adapt the construction to the post-quantum regime. Natural attempts to do so run into significant issues that need several new ideas (and a new hardness assumption) to resolve.

2.3 Post-Quantum TLP from Isogenies

Challenges to Post-Quantum Extension. As illustrated by [Fig. 1](#), our classical TLP design is inherently based on a commutative diagram: the isogeny computation and the scalar multiplication must commute for the protocol to be correct. In cryptographic terms, a commutative diagram (with a suitable hardness assumption) is a one-round key exchange: there is an implicit key exchange in our TLP design where E, P are public parameters, E', P' are the public key of one party and Q is the public key of the second party, and Q' is the shared secret. We can thus view this TLP as an instantiation of a more generic framework: any one-round key exchange where one party's computations are sequential gives rise to a TLP. This key exchange interpolates between ECDH and an isogeny-based key exchange, providing post-quantum security for one party. Thus, a natural attempt would be to somehow replace the quantum-insecure scalar multiplication with some quantum-secure alternative.

A possible approach would instantiate this key exchange from CSIDH: the scalar multiplication $[x]$ is replaced by an $\mathbb{F}_p$ isogeny, and the isogeny ϕ is also chosen to be $\mathbb{F}_p$ -rational. In this way, the two isogenies commute. Given a representation of ϕ in the public parameters pp_{solve} , the solver could compute ϕ' without any additional information, which gives us a compact puzzle description. However, the recent advances in the computation of $\mathbb{F}_p$ -isogenies [\[28,19\]](#) have shown that computing $\mathbb{F}_p$ -isogenies can be efficient, *regardless of the degree of the isogeny*. This breaks the sequentiality assumption for $\mathbb{F}_p$ -isogenies, and indeed it has broken several timed constructions from $\mathbb{F}_p$ -isogenies, such as the VDFs from [\[21\]](#) and the first construction in [\[20\]](#). The algorithm for efficient computation of $\mathbb{F}_p$ -isogenies similarly applies to isogenies between ordinary curves [\[33\]](#), which thus cannot offer a promising direction for a post-quantum TLP with setup.

Our Approach: Relying on Pushforward Isogeny Computations. Consider the following alternative approach: we keep ϕ as an $\mathbb{F}_{p^2}$ -isogeny, but replace multiplication by $[x]$ with a second $\mathbb{F}_{p^2}$ -isogeny ψ whose degree is coprime to ℓ . The puzzle contains $\psi : E \rightarrow F$. The puzzle solver computes the codomain F' of the pushforward $[\phi]_*\psi : E' \rightarrow F'$. The puzzle generator uses a trapdoor

$$\begin{array}{ccc}
E & \xrightarrow{\phi_T} & E' \\
\psi \downarrow & & \downarrow [\phi_T]_* \psi \\
F & \xrightarrow{[\psi]_* \phi_T} & F'
\end{array}$$

Fig. 2. Illustration of pushforwards of isogenies ϕ_T and ψ of coprime degree.

shortcut for ϕ to compute the same codomain efficiently. We realize this shortcut by relying on *pushforward isogeny computations*. Pushforwards are central to isogeny-based cryptography, and are used by many isogeny-based PKE schemes. Examples include CSIDH [16], M-SIDH [22], binSIDH [7], and POKÉ [8]. Suppose that ϕ and ψ have coprime degrees. The two pushforwards then have the same codomain, and hence, we have $[\phi]_* \psi \circ \phi = [\psi]_* \phi \circ \psi$. This yields the commutative diagram in Fig. 2.

This gives hope – suppose we modify the classical TLP with setup by replacing point evaluation with a pushforward computation as follows: given a long ℓ^T -isogeny ϕ_T ¹ and an isogeny ψ , the puzzle solver should need to compute the codomain F' of the pushforward $[\phi_T]_* \psi$ in order to recover the secret. The computation of ϕ_T can be conjectured to be sequential in T (please see Section 5.4 for a detailed discussion), so the hope is that the solver is forced to compute ϕ_T to recover the secret.

Attempt-1. A natural approach would be to include F in the puzzle description, and ask the solver to compute the (codomain F' of the) pushforward isogeny $[\phi_T]_* \psi$ (or, equivalently of $[\psi]_* \phi_T$). However, this does not work since given just the curve F , with no torsion information, it is hard to compute the pushforward. The solution in SIDH-based schemes [22,7,8] is to include torsion points on F to enable the computation of $[\psi]_* \phi_T$. However, the order of the additional torsion points must be the same as the degree of ϕ_T , which is ℓ^T . This means that the size of the puzzle description (that includes F and the additional torsion points) grows with T , making the puzzle not succinct, which is required by definition.

Attempt-2. A possible way forward is to change the puzzle definition to include the entire isogeny $\psi : E \rightarrow F$, rather than just its codomain. Since the solver has access to both ψ and ϕ_T , it can now compute a pushforward isogeny and hence the codomain F' . But this introduces another issue: the puzzle solver can now potentially solve the puzzle in time independent of T . If ψ has a smooth degree, then the solver could precompute the action ϕ_T on the $(\deg \psi)$ -torsion; then, given a ψ , they can immediately obtain the kernel of $[\phi_T]_* \psi$, from which they can compute F' by computing the isogeny $[\phi_T]_* \psi$.

¹ The subscript T is used to emphasize dependence on T .

Attempt-3. An idea to overcome the above issue would be to choose ψ to have a large *prime* degree $N = \Theta(2^{2\lambda})$. Even if the above pre-computation provides the kernel of $[\phi_T]_*\psi$, computing its codomain still requires a large prime-degree isogeny computation. While this approach gives us a puzzle that requires sequential work to solve, it is still unclear how to make the puzzle generation efficient. Currently, without knowing the endomorphism ring of E , the puzzle generator cannot efficiently sample the prime-degree isogeny ψ ; moreover, given ψ , the puzzle generator also has to compute $[\phi_T]_*\psi$, which again takes time proportional to T .

Can We Use Trapdoors? Suppose that the puzzle generator knows the endomorphism ring of E . It can then sample a random large-prime-degree isogeny $\psi : E \rightarrow F$ in time independent of T . Also suppose that it knows a short isogeny $\phi_{\text{short}} : E \rightarrow E'$. Its degree must be independent of T . Its action on $E[N]$ must agree with that of ϕ_T up to an invertible scalar. The pushforwards $[\phi_{\text{short}}]_*\psi$ and $[\phi_T]_*\psi$ then have the same kernel and codomain. Thus, the puzzle generator can compute F' efficiently.

To enable this, we introduce an efficient technique (see [Section 5.2](#)) to find ϕ_{short} given ϕ_T and the endomorphism ring of E . This technique, however, requires evaluating ϕ_T , which means it takes time $O(T)$. To bypass this, we should ideally have setup sample ϕ_{short} , and then share a trapdoor $\text{td} = (\text{End}(E), \phi_{\text{short}})$ “symmetrically” with the puzzle generator, such that it is not available to the attacker. This suggests a weaker notion of *symmetric* TLP in the preprocessing model – namely, a model where the setup and puzzle generation algorithms share a secret trapdoor (reusable across multiple puzzle generation instances). However, this seems useless – a puzzle generation which needs a secret key is not meaningful for TLPs in general.

Symmetric TLP with Setup Suffices! At this point, we make a crucial observation: our compiler from preprocessing-based TLP to TLP without preprocessing can, in fact, be weakened to begin with a *symmetric* TLP in the preprocessing model as outlined above. We formally define such a notion of symmetric TLP with preprocessing/setup in [Section 4.1](#). While such a preprocessing model is completely useless for any practical purposes, it suffices nicely for our compiler, since the setup algorithm in the preprocessing model is executed *inside* the puzzle generation algorithm of TLP without setup. We formally present the compiler from symmetric TLP with (long) setup to TLP without setup with $\sqrt{T}$ efficiency in [Section 4.2](#).

Isogeny-based Post-Quantum Symmetric TLP with Setup. We now describe a construction of post-quantum symmetric TLP with setup, based on a sequentiality assumption for pushforward computations. Applying our compiler in [Section 4.2](#) to this construction yields the first isogeny-based post-quantum TLP without setup that achieves $\sqrt{T}$ puzzle-generation efficiency. The construction is outlined below and is also succinctly represented as [Fig. 3](#). Throughout, H denotes

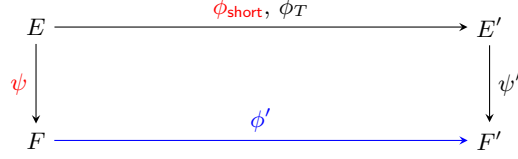


Fig. 3. Our post-quantum symmetric TLP with setup based on isogeny pushforwards. The red terms need a trapdoor for their generation. The puzzle generator computes F' with $[\phi_{\text{short}}]_*\psi$. The solver computes F' with $[\phi_T]_*\psi$. The blue edge shows an adversarially attempted shortcut that fails because ψ has exponentially large degree.

a suitable hash function (modeled as a random oracle). Please see [Section 5.1](#) for the detailed construction.

Setup. Setup samples a supersingular curve $E/\mathbb{F}_{p^2}$, retains its endomorphism ring, and samples a public seed $\text{seed}_\phi \leftarrow \{0, 1\}^\lambda$. It computes the seeded deterministic degree- ℓ^T isogeny $\phi_T = \text{DetIsog}(E, \ell, T; \text{seed}_\phi) : E \rightarrow E'$ using [Algorithm 1](#). It then uses $\text{End}(E)$ to compute a short isogeny $\phi_{\text{short}} : E \rightarrow E'$ and an invertible scalar $\mu \in \mathbb{Z}_N^\times$ (using a procedure described in [Algorithm 2](#)) such that $\phi_{\text{short}}(P) = [\mu]\phi_T(P)$ for every $P \in E[N]$. The degree of ϕ_{short} is polynomial in p and N , and is independent of T . Let $\text{td} = (\text{End}(E), \phi_{\text{short}})$. Setup outputs

$$\text{st} = (\text{td}, \text{pp}), \quad \text{pp} = ((p, N, \ell, T), (E, E'), \text{seed}_\phi),$$

where $p = \Theta(2^{3\lambda})$, $N = \Theta(2^{2\lambda})$, and $\ell = O(1)$. The seed is a succinct public description of the long walk. The state st , which contains the trapdoor td , is only shared between setup and puzzle generation, while the puzzle solver is only given the public parameters pp .

Puzzle Generation. Given a solution $s \in \{0, 1\}^\lambda$, the puzzle generator samples a random degree- N isogeny $\psi : E \rightarrow F$, where N is a large prime. It uses ϕ_{short} to compute the codomain F' of $[\phi_{\text{short}}]_*\psi$, and outputs the puzzle

$$Z = (\text{pp}, \psi, H(j(F')) \oplus s),$$

where the hash computation uses the canonical encoding of $j(F')$.

Since the puzzle generator knows $\text{End}(E)$, the sampling of ψ is efficient, and independent of T . Also, since the actions of ϕ_{short} and ϕ_T on $E[N]$ differ only by the invertible scalar μ , they map $\ker \psi$ to the same subgroup. Thus $[\phi_{\text{short}}]_*\psi$ and $[\phi_T]_*\psi$ have the same codomain up to isomorphism. Finally, the computation of $[\phi_{\text{short}}]_*\psi$ is efficient, and independent of T . Thus, puzzle generation is efficient overall. Observe that we crucially rely on the *symmetric* nature of the TLP with setup here.

Puzzle Solving. The puzzle gives ψ to the solver, and the public parameters give (E, seed_ϕ, T) . The solver reconstructs $\phi_T = \text{DetIsog}(E, \ell, T; \text{seed}_\phi)$ and computes the codomain F' of $[\phi_T]_*\psi$ sequentially. It then computes $H(j(F'))$ to recover s .

Correctness follows immediately. See [Section 5.3](#) for a detailed discussion on parameter choices and efficiency. To argue the security of the above symmetric TLP with setup construction, we rely on the conjectured (post-quantum) sequentiality of pushforward isogeny computations, stated informally below.

Assumption: Sequentiality of Pushforward Computations. Given a security parameter λ and a sequentiality parameter $T = \text{poly}(\lambda)$: let $p = \Theta(2^{3\lambda})$ and $N = \Theta(2^{2\lambda})$ be primes, and let $\ell = O(1)$ be a small prime. Let $E/\mathbb{F}_{p^2}$ be a supersingular elliptic curve with unknown endomorphism ring. Let $\phi_T \rightarrow E'$ be a random cyclic degree- ℓ^T isogeny, represented as a length- T sequence of degree- ℓ isogenies. Also, let $\{\psi_i\}_{i \in [q]}$ be independently and uniformly sampled degree- N isogenies represented in higher-dimensional (HD) form. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be a two-phase quantum adversary. $\mathcal{A}_1$ preprocesses $(\phi_T, \{\psi_i\}_{i \in [q]})$ without any depth-restriction and outputs some state st . After this preprocessing is complete, the challenger samples a fresh and independent degree- N isogeny $\psi \rightarrow F$ and gives (st, ψ) to $\mathcal{A}_2$. Let F' denote the codomain obtained by pushing ψ through ϕ_T . The assumption states that if $\mathcal{A}_2$ has parallel running time $o(T)$, then its probability of outputting F' is negligible.

Informally speaking, the above assumption captures more than just the hardness of computing a pushforward codomain corresponding to a fresh, uniformly random ψ . It posits that even extensive preprocessing tailored to the fixed long isogeny ϕ_T (including processing polynomially many other degree- N isogenies) does not enable the adversary to compute the pushforward codomain of a fresh random isogeny ψ in $o(T)$ parallel time. Although the protocol publishes only the seeded description (E, seed_ϕ, T) , the assumption gives the preprocessing phase the full long isogeny. This two-stage, multi-use formulation of the assumption is necessary for arguing reusable post-quantum security of our symmetric TLP with setup. See [Assumption 1 \(Section 5.4\)](#) for a more formal description, and [Section 5.4](#) for a detailed justification.

Proving Reusable Symmetric Security. We prove the post-quantum reusable security of our symmetric TLP with setup based on the above assumption in the QROM. We first provide an informal overview of the proof in the classical ROM, and then discuss how to extend it to the QROM.

For the classical ROM-based proof, in the preprocessing phase, the reduction uses the pre-challenge isogenies ψ_i supplied by the assumption to simulate all of the adversary's puzzle-generation queries; since this phase is not depth bounded, it can compute the corresponding pushforward codomains directly. It also programs the random oracle defining the public isogeny walk before any adversarial query so that the public seed corresponds to the long isogeny supplied by the assumption. When the fresh challenge isogeny ψ arrives, however, the reduction cannot compute its pushforward codomain F' within $o(T)$ depth. It therefore replaces the value $H_{\text{pf}}(F')$ used to mask the challenge solution by an independent random value u , in which case the challenge bit is perfectly hidden. Consequently, any non-negligible advantage in distinguishing the real puzzle from this perfectly hiding simulation must reveal information about the missing point

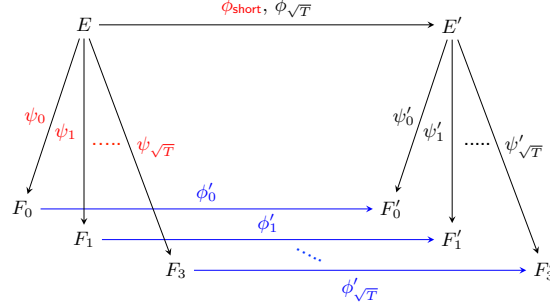


Fig. 4. Isogenies used to generate $\sqrt{T}$ puzzles in the compiled post-quantum TLP without external setup. The red terms need a trapdoor for their generation. The blue edges show adversarially attempted shortcuts that fail because each ψ_i has high degree. We prove the compiled construction secure in the QROM.

F' . In the classical ROM, this means that the adversary must explicitly query H_{pf} on F' with non-negligible probability; the reduction records the adversary's oracle queries and outputs a uniformly chosen queried curve, recovering F' with only a polynomial loss. See [Theorem 4 \(Section 5.4\)](#) and its proof for more details.

In the QROM version of the proof, individual queries cannot be observed without disturbing the computation. So, we instead use a one-point QROM switching/O2H argument: a non-negligible distinguishing advantage implies noticeable total query mass on F' , and measuring a uniformly selected query register extracts F' with non-negligible probability. Importantly, this reduction transfers the adversary's quantum state only once, never rewinds it, and preserves the $o(T)$ online depth bound, thereby contradicting the above assumption. See [Theorem 5 \(Appendix A.7\)](#) and its proof for more details.

Putting Everything Together. We finally compile the above symmetric TLP with setup into a TLP without setup having $\sqrt{T}$ puzzle-generation efficiency following the compilation strategy outlined in [Section 2.1](#).

Concretely, the compiled puzzle generation first runs the setup procedure of our symmetric TLP with setup using sequentiality parameter $\sqrt{T}$ to generate an isogeny $\phi_{\sqrt{T}} : E \rightarrow E'$, along with the trapdoor information, consisting of $\text{End}(E)$ and an equivalent short representation of $\phi_{\sqrt{T}}$, namely ϕ_{short} . This takes time $O(\sqrt{T})$. It then uses the trapdoor td to sample $\sqrt{T}$ fresh puzzles (see [Fig. 4](#) for an illustration). The intermediate puzzles encode independent keys k_i . The mask $H(k_i)$ hides each later puzzle until the solver solves the preceding puzzle. Puzzle generation in the symmetric TLP with setup is independent of T . Therefore, this stage takes $O(\sqrt{T})$ time.

As described in [Section 2.1](#), each $H(k_i)$ masks the next puzzle and forces the solver to traverse the resulting chain sequentially. Thus, solving the complete chain takes $T \cdot \text{poly}(\lambda)$ time.

The resulting construction has no external setup or preprocessing: the trapdoor is generated internally by the compiled puzzle generator and never appears in the final TLP interface. Assuming that the underlying isogeny TLP satisfies post-quantum reusable security in the QROM ([Theorem 5](#) under [Assumption 1](#)), and applying the QROM compiler theorem ([Theorem 6](#)) therefore yields a TLP without setup under [Assumption 1](#) in the QROM with $\sqrt{T} \cdot \text{poly}(\log T, \lambda)$ puzzle-generation time and $T \cdot \text{poly}(\lambda)$ puzzle-solving time.

2.4 Proof-of-Concept Implementation

We benchmark a SageMath-based prototype implementation of the compiled plain TLP construction. We use SQIsign level-I-motivated parameters, a two-dimensional representation of the large-degree isogenies, and high-degree chunks for the deterministic walk. For each target delay, we optimize T_0 and T_1 for generation time and generate the T_1 underlying puzzles in parallel before linking them. The results provide an end-to-end validation of the complete construction, demonstrate that the theoretical generation–solving separation translates into a meaningful concrete gap, and quantify the resulting tradeoff between puzzle-generation time and puzzle size. See [Section 6](#) for the implementation details, parameters, and benchmarking results.

3 Preliminaries

Notation. For $n > 0$, let $[n] = \{1, \dots, n\}$. For integers $m \leq n$, let $[m, n] = \{m, \dots, n\}$. We use λ for the security parameter. The symbols $\text{poly}(\lambda)$ and $\text{negl}(\lambda)$ denote unspecified polynomial and negligible functions. If S is a finite set, then $s \leftarrow S$ samples s uniformly from S .

3.1 Plain Time-Lock Puzzle (TLP) without Setup/Preprocessing

We first recall the standard definition of a plain time-lock puzzle (TLP) (i.e., without setup/preprocessing) [\[11\]](#).

Definition 1 (Time-Lock Puzzles). *A time-lock puzzle is a tuple of two algorithms (PGen, PSolve) defined as follows.*

- $Z \leftarrow \text{PGen}(1^\lambda, T(\lambda), s)$: a probabilistic algorithm that takes as input a security parameter λ , a sequentiality parameter T , and a solution $s \in \{0, 1\}^\lambda$, and outputs a puzzle Z .
- $s \leftarrow \text{PSolve}(Z)$: a deterministic algorithm that takes as input a puzzle Z and outputs a solution s .

Remark 1. We implicitly assume that Z contain descriptions of the security parameter λ and the sequentiality parameter T , and so these are not mentioned as explicit inputs to PSolve .

Definition 2 (Correctness). For all $\lambda \in \mathbb{N}$, for all polynomials T in λ , and for all $s \in \{0, 1\}^\lambda$, it holds that $s = \text{PSolve}(\text{PGen}(\lambda, T, s))$.

Definition 3 (Security). A scheme $(\text{PGen}, \text{PSolve})$ is secure if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $o(T)$, there exists a negligible function $\mu(\cdot)$, such that for any security parameter $\lambda \in \mathbb{N}$ it holds that

$$\Pr \left[\begin{array}{l} (\text{st}, s_0, s_1) \leftarrow \mathcal{A}_1(1^\lambda, T(\lambda)), \quad s_0, s_1 \in \{0, 1\}^\lambda \\ b \leftarrow \mathcal{A}_2(Z, \text{st}) : b \leftarrow \{0, 1\} \\ Z \leftarrow \text{PGen}(1^\lambda, T(\lambda), s_b) \end{array} \right] \leq \frac{1}{2} + \mu(\lambda).$$

Remark 2. Throughout this paper, following the standard parallel-time convention, the *depth* of an adversary is measured in *rounds*: each round comprises at most $p_0(\lambda)$ sequential gates (for a fixed polynomial p_0), and at most one layer of parallel oracle calls (e.g., in the (Q)ROM). A depth- d adversary thus has total gate-depth at most $d \cdot p_0(\lambda)$.

Efficiency. We say that a TLP is *ideally efficient* if the time to compute the puzzle is $\text{poly}(\log T, \lambda)$ (i.e., polylogarithmic in the sequentiality parameter T). If puzzle generation takes time $T^\epsilon \cdot \text{poly}(\log T, \lambda)$ for some constant $\epsilon \in [0, 1]$, we (informally) refer to it as a *TLP with T^ϵ efficiency*.

3.2 TLP with Setup/Preprocessing

We now recall the definition of a TLP with setup (equivalently, a TLP in the *preprocessing model*). Such a TLP has the same syntax as a plain TLP as in [Definition 1](#), except that it has the following additional algorithm:

- $\text{pp} \leftarrow \text{PSetup}(1^\lambda, T(\lambda))$: a probabilistic algorithm that takes as input a security parameter 1^λ and a time hardness parameter T , and outputs public parameters pp . We assume pp is input to the remaining algorithms.

We then modify the security notion accordingly.

Definition 4 (Reusable Security). A scheme $(\text{PSetup}, \text{PGen}, \text{PSolve})$ is reusable secure if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $o(T)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{PSetup}(1^\lambda, T(\lambda)) \\ b \leftarrow \mathcal{A}_2(Z, \text{st}) : \begin{array}{l} (\text{st}, s_0, s_1) \leftarrow \mathcal{A}_1(\text{pp}), \quad s_0, s_1 \in \{0, 1\}^\lambda \\ b \leftarrow \{0, 1\} \\ Z \leftarrow \text{PGen}(\text{pp}, s_b) \end{array} \end{array} \right] \leq \frac{1}{2} + \mu(\lambda).$$

3.3 Isogeny Graphs over $\mathbb{F}_{p^2}$

For an elliptic curve $E/\mathbb{F}_q$, define $E[N] = \{P \in E(\overline{\mathbb{F}}_q) : [N]P = O\}$, where O is the identity. Thus, $E[N]$ contains the points whose orders divide N . Let p be the characteristic of $\mathbb{F}_q$. We call E supersingular when $E[p] = \{O\}$.

Each supersingular j -invariant has a representative over $\mathbb{F}_{p^2}$. In this work, we always use a representative $E/\mathbb{F}_{p^2}$ that satisfies $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. If $N \mid p+1$, then $E[N] \subseteq E(\mathbb{F}_{p^2})$ and $E[N] \cong \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$. In this case, standard algorithms can sample a point of exact order N or a basis of $E[N]$ in expected polynomial time [31, Lemma 6.1].

An *isogeny* $\phi : E \rightarrow E'$ is a nonconstant rational map that sends the identity of E to the identity of E' . Every isogeny is a group homomorphism; in particular,

$$\phi([a]P + [b]Q) = [a]\phi(P) + [b]\phi(Q)$$

for all points P, Q and integers a, b . Throughout this work, all isogenies are separable, so their *degree* equals the cardinality of their kernel: $\deg \phi = |\ker \phi|$. If $\deg \phi$ is coprime to the order of a point P , then ϕ preserves its order, i.e., $\text{ord}(\phi(P)) = \text{ord}(P)$. Degrees are multiplicative under composition: $\deg(\psi \circ \phi) = \deg \psi \cdot \deg \phi$. Every isogeny of composite degree decomposes into a sequence of isogenies of prime degree. In particular, an isogeny of degree ℓ^T , for a prime ℓ , can be written as a composition of T isogenies of degree ℓ . An isogeny is *cyclic* if its kernel is cyclic. Such an isogeny cannot factor through a scalar multiplication map $[m]$ with $|m| > 1$. Any isogeny can be represented efficiently via a higher-dimensional (HD) representation [32].

For a prime $\ell \neq p$, the supersingular ℓ -isogeny graph has one vertex for each supersingular j -invariant in $\mathbb{F}_{p^2}$, with edges representing ℓ -isogenies. It has $p/12 + O(1)$ vertices, each with $\ell+1$ outgoing edges (counted with multiplicity). Its Ramanujan property ensures rapid convergence of random walks to the stationary distribution [6]. This distribution is within statistical distance $O(1/p)$ of uniform, with the discrepancy arising from curves with extra automorphisms. Thus, for the parameters used in this work, sufficiently long walks allow polynomial-time sampling of curves whose distribution is negligibly close to uniform. A non-backtracking walk of length T corresponds to a cyclic isogeny of degree ℓ^T .

4 TLP without Setup from Symmetric TLP with Setup

In this section, we present a generic compiler that, given a symmetric TLP with setup and a hash function modeled as a random oracle, yields a plain TLP without setup/preprocessing as per Definition 1. This symmetric notion is weaker than the usual TLP-with-setup interface and is defined below.

4.1 Symmetric TLP with Setup

We adapt the definition of TLP with setup (Definition 4) to a weaker *symmetric* setting below.

Definition 5 (Symmetric TLP with Setup). A symmetric time-lock puzzle with setup is a tuple of three algorithms (PSetup, PGen, PSolve) defined as follows.

- $\text{st} \leftarrow \text{PSetup}(1^\lambda, T(\lambda))$: a probabilistic algorithm that takes as input a security parameter $\lambda \in \mathbb{N}$ and a sequentiality parameter T , and outputs a secret state st used for puzzle generation.
- $Z \leftarrow \text{PGen}(\text{st}, s)$: a probabilistic algorithm that takes as input the secret state st and a solution $s \in \{0, 1\}^\lambda$, and outputs a puzzle Z .
- $s \leftarrow \text{PSolve}(Z)$: a deterministic algorithm that takes as input the puzzle Z , and outputs a solution $s \in \{0, 1\}^\lambda$.

Remark 3. We implicitly assume that st and Z contain descriptions of the security parameter λ and the sequentiality parameter T , and so these are not mentioned as explicit inputs to PGen and PSolve.

Definition 6 (Correctness of Symmetric TLP with Setup). We require that for any security parameter $\lambda \in \mathbb{N}$, any sequentiality parameter T , any $s \in \{0, 1\}^\lambda$, and any $\text{st} \leftarrow \text{PSetup}(1^\lambda, T)$, it holds that

$$\text{PSolve}(\text{PGen}(\text{st}, s)) = s.$$

Definition 7 (Symmetric Reusable Security). A scheme (PSetup, PGen, PSolve) is reusable secure if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $o(T)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that

$$\Pr \left[\begin{array}{l} \text{st} \leftarrow \text{PSetup}(1^\lambda, T(\lambda)) \\ b \leftarrow \mathcal{A}_2(Z, \mathbf{z}) : \begin{array}{l} (\mathbf{z}, s_0, s_1) \leftarrow \mathcal{A}_1^\mathcal{O}(1^\lambda, T(\lambda)), \quad s_0, s_1 \in \{0, 1\}^\lambda \\ b \leftarrow \{0, 1\} \\ Z \leftarrow \text{PGen}(\text{st}, s_b) \end{array} \end{array} \right] \leq \frac{1}{2} + \mu(\lambda).$$

where $\mathcal{O} = \text{PGen}(\text{st}, \cdot)$ is the puzzle generation oracle which $\mathcal{A}_1$ can query with solutions of its choice.

Remark 4. In the above definition, the state st is *read-only* in the sense that, while $\mathcal{O} = \text{PGen}(\text{st}, \cdot)$ may reuse st across arbitrarily many puzzle-generation queries, it may not modify st . Also, each invocation of $\mathcal{O} = \text{PGen}(\text{st}, \cdot)$ uses fresh, independent randomness.

Definition 8 (Efficiency of Symmetric TLP with Setup). We require that for any security parameter $\lambda \in \mathbb{N}$, any sequentiality parameter T , and any $\text{st} \leftarrow \text{PSetup}(1^\lambda, T)$:

- $\text{st} \leftarrow \text{PSetup}(1^\lambda, T)$ can be computed in time $T \cdot \text{poly}(\lambda)$.
- $Z \leftarrow \text{PGen}(\text{st}, s)$ can be computed in time $\text{poly}(\log T, \lambda)$ for any $s \in \{0, 1\}^\lambda$.
- $s \leftarrow \text{PSolve}(Z)$ can be computed in time $O(T \cdot \text{poly}(\lambda))$.

4.2 Symmetric TLP with Setup to Plain TLP without Setup

In this subsection, we present a compiler that bootstraps any generic symmetric TLP with setup into a plain TLP without setup/preprocessing. The compiler is parameterized by two quantities: T_0 , the sequentiality parameter used for the underlying TLP, and T_1 , the number of underlying puzzles used by the compiler. We take $T = T_0 T_1$. The balanced choice $T_0 = T_1 = \sqrt{T}$ gives the claimed $\sqrt{T}$ efficiency for puzzle generation. Our construction additionally uses a hash function H , whose output length is set as needed (we model H as a random oracle in the proof). We identify puzzle descriptions with their canonical bit-string encodings whenever they are XORed with outputs of H . We present the compiler below.

The Compiler. Given any symmetric TLP with setup ($\text{PSetup}, \text{PGen}, \text{PSolve}$) for λ -bit solutions, the compiler outputs a TLP without setup ($\text{PGen}^*, \text{PSolve}^*$) for λ -bit solutions as follows.

$\text{PGen}^*(\lambda, T, s)$: On input the security parameter λ , the sequentiality parameter T and the secret $s \in \{0, 1\}^\lambda$, the puzzle generation for the TLP without setup proceeds as follows:

1. Run the PSetup sub-routine of the original symmetric TLP with setup internally using sequentiality parameter T_0 as $\text{st} \leftarrow \text{PSetup}(1^\lambda, T_0)$.
2. Set $s_{T_1} = s$. For each $i \in [1, T_1 - 1]$, sample a fresh intermediate key $k_i \leftarrow \{0, 1\}^\lambda$ and set $s_i = k_i$.
3. For each $i \in [1, T_1]$, compute $Z_i = \text{PGen}(\text{st}, s_i)$.
4. For each $i \in [2, T_1]$, compute $\text{ct}_i = Z_i \oplus H(k_{i-1})$.
5. Eventually, output the puzzle $Z^* = (Z_1, \text{ct}_2, \dots, \text{ct}_{T_1})$.

$\text{PSolve}^*(Z^*)$: The puzzle solver for the TLP without setup takes as input a puzzle $Z^* = (Z_1, \text{ct}_2, \dots, \text{ct}_{T_1})$, and does the following for each $i = 1, \dots, T_1 - 1$ in sequence:

1. Compute $k_i = \text{PSolve}(Z_i)$.
2. Recover Z_{i+1} as $Z_{i+1} = \text{ct}_{i+1} \oplus H(k_i)$.
3. Finally, output the solution $s = \text{PSolve}(Z_{T_1})$.

We state and prove the following theorem for the above compiler.

Theorem 1 (Our Compiler in the ROM). *Under the assumption that $(\text{PSetup}, \text{PGen}, \text{PSolve})$ is a symmetric TLP with setup as per Definition 5 that satisfies correctness as in Definition 6, symmetric reusable security as in Definition 7, and efficiency as in Definition 8 for sequentiality parameter T_0 , and under the assumption that H is a random oracle, the above compiler yields a plain TLP without setup as per Definition 1 that satisfies correctness as in Definition 2, security as in Definition 3, and has puzzle generation time $(T_0 + T_1) \cdot \text{poly}(\log T, \lambda)$, and puzzle solving time $T_0 T_1 \cdot \text{poly}(\lambda) = T \cdot \text{poly}(\lambda)$. In particular, the balanced choice $T_0 = T_1 = \sqrt{T}$ gives $\sqrt{T}$ efficiency for puzzle generation.*

Correctness. Correctness of the compiled TLP without setup ($\text{PGen}^*, \text{PSolve}^*$) follows immediately from the correctness of the original TLP with setup. Each puzzle Z_i for $i < T_1$ yields the key needed to decrypt ct_{i+1} and recover the next puzzle Z_{i+1} . The last puzzle Z_{T_1} yields the desired solution s .

Efficiency. PGen^* first runs the setup of the original symmetric TLP with sequentiality parameter T_0 , which takes time $T_0 \cdot \text{poly}(\lambda)$. It then generates T_1 puzzle instances, each in time $\text{poly}(\log T_0, \lambda)$, and does additional polynomial-time work to sample the intermediate keys and compute the ciphertexts. Overall, puzzle generation takes time $(T_0 + T_1) \cdot \text{poly}(\log T, \lambda)$. For the balanced choice $T_0 = T_1 = \sqrt{T}$, this is $\sqrt{T} \cdot \text{poly}(\log T, \lambda) = \sqrt{T} \cdot \text{poly}(\lambda)$, for polynomially bounded T .

Finally, PSolve^* solves T_1 puzzle instances, each with sequentiality parameter T_0 , and does some additional $\text{poly}(\lambda)$ work at each step to recover the next puzzle. This takes time $T_0 T_1 \cdot \text{poly}(\lambda)$, as desired. This completes the efficiency analysis for the compiler.

Security. Let λ be the security parameter. Assume, for contradiction, that there exists a polynomial-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with the parallel depth of $\mathcal{A}_2$ being $d = o(T) = o(T_0 T_1)$, such that $\mathcal{A}$ breaks the security of the compiled, TLP without setup/preprocessing with non-negligible advantage $\text{Adv}_{\text{plain}}(\mathcal{A})$, i.e., achieves success probability $1/2 + \text{Adv}_{\text{plain}}(\mathcal{A})$ in the security experiment described in [Definition 3](#). Let s_0, s_1 be the two challenge solutions chosen by $\mathcal{A}$ in this experiment. Also, let $q_H = \text{poly}(\lambda)$ be an upper bound on the number of random-oracle queries made by $\mathcal{A}$. We assume that every canonical puzzle encoding is padded to a fixed length $m_Z = m_Z(\lambda, T_0)$. Thus, the random oracle has range $\{0, 1\}^{m_Z}$ in this proof. In the TLP without setup experiment, for a challenge bit $b \in \{0, 1\}$, the adversary $\mathcal{A}$ receives a challenge puzzle of the form $Z^* = (Z_1, \text{ct}_2, \dots, \text{ct}_{T_1})$, where, for uniformly random and independently sampled intermediate keys $k_1, \dots, k_{T_1-1} \leftarrow \{0, 1\}^\lambda$,

$$Z_i \leftarrow \text{PGen}(\text{st}, k_i) \text{ for each } i < T_1, \quad Z_{T_1} \leftarrow \text{PGen}(\text{st}, s_b),$$

and where, for each $i \in [T_1 - 1]$, $\text{ct}_{i+1} = Z_{i+1} \oplus H(k_i)$, where the underlying symmetric TLP with setup ($\text{PSetup}, \text{PGen}, \text{PSolve}$) uses sequentiality parameter T_0 . We define Bad as the union of the following events:

- (i) $\mathcal{A}_1$ queries an intermediate key before the challenge puzzle is generated, or $\mathcal{A}_2$ queries k_i before Z_i is revealed (note that the puzzle Z_1 is revealed when $\mathcal{A}_2$ receives the compiled challenge).
- (ii) two independently sampled intermediate keys collide, i.e., $k_i = k_j$ for some $i, j \in [T_1 - 1]$ such that $i \neq j$ (recall that $k_1, \dots, k_{T_1-1}$ are sampled as intermediate keys, while the last solution slot carries the actual puzzle solution s_b).

Remark 5. For $i \geq 2$, puzzle Z_i is revealed when the first query $H(k_{i-1})$ receives its answer. Thus, a query to k_i in the same depth layer as the first query to k_{i-1} occurs before Z_i is revealed.

We also denote the complementary event as $\text{Good} = \neg \text{Bad}$. We now analyze the TLP-without-setup security experiment conditioned on Good .

Lemma 1. *Conditioned on the event Good , unless the adversary $\mathcal{A}$ queries the random oracle on every intermediate key $k_1, \dots, k_{T_1-1}$ in order, its view is independent of the challenge bit b .*

Proof. Fix some $i \in [T_1 - 1]$. Before the query k_i is made to the random oracle, the value $H(k_i)$ is a fresh uniform string and behaves as a perfect one-time pad for Z_{i+1} , thereby revealing no information about it. This follows immediately from the property of a random oracle. Moreover, conditioned on the event Good , $\mathcal{A}$ cannot query the random oracle on k_i before Z_i is revealed. Thus the first instance of the query $H(k_i)$ occurs only after Z_i has become available. By an inductive argument, the entire subset of puzzles $\{Z_{i+1}, \dots, Z_{T_1}\}$ is hidden from the view of $\mathcal{A}$.

Formally, on executions where Good holds and some k_i ($i \leq T_1 - 1$) is never queried, the adversary $\mathcal{A}$'s view in the experiment is identically distributed for $b = 0$ and $b = 1$, because the only b -dependent component of the view, Z_{T_1} inside ct_{T_1} , is masked by a one-time pad that remains fresh. Consequently, the following holds information-theoretically:

$$\Pr[b' = b \wedge \text{Good} \wedge \neg \text{AllKeysSeq}] \leq \frac{1}{2} \Pr[\text{Good} \wedge \neg \text{AllKeysSeq}],$$

where b' denotes $\mathcal{A}$'s output. In other words, $\mathcal{A}$'s view is independent of b . This completes the proof of [Lemma 1](#).

Lemma 2. *In the security experiment for the compiled TLP without setup, we have $\Pr[\text{Bad}] \leq \text{negl}(\lambda)$.*

Proof. Since each k_i is uniformly random and independently sampled, at any time before Z_i is revealed to $\mathcal{A}$, the key k_i is a uniform λ -bit string independent of the $\mathcal{A}$'s view. Hence

$$\Pr[\text{Bad}] \leq \frac{(T_1 - 1)q_H}{2^\lambda} + \frac{(T_1 - 1)^2}{2^{\lambda+1}} = \text{negl}(\lambda).$$

We explain the above expression in a bit more detail:

- For the first term, fix an intermediate key k_i and one query made before Z_i is revealed. This includes queries by $\mathcal{A}_1$. Conditioned on no earlier bad event, k_i is uniform and independent of the query. The query equals k_i with probability $2^{-\lambda}$. A union bound over at most q_H queries and $T_1 - 1$ keys gives the first term.
- The second term is a union bound over at most $\binom{T_1-1}{2} \leq (T_1 - 1)^2/2$ pairs of independent uniform keys, each pair colliding with probability $2^{-\lambda}$. We remark that the phrase “before Z_i is revealed” implicitly relies on the layers above i being masked as well; formally, by a joint induction over the queries of $\mathcal{A}$, as long as Z_i is not revealed, the masks of all unrevealed layers are fresh,

hence (by the argument in the proof of [Lemma 1](#)) the keys of all unrevealed layers are uniform and independent of the view, and the next query is a fresh guess for each of them.

This completes the proof of [Lemma 2](#).

It follows that the event **Good** happens with overwhelmingly large probability.

Next, let **AllKeysSeq** denote the event that the adversary $\mathcal{A}$ issues all relevant random oracle queries on $k_1, \dots, k_{T_1-1}$ sequentially in the order

$$H(k_1), H(k_2), \dots, H(k_{T_1-1}).$$

[Lemma 1](#) and [Lemma 2](#) together imply that, except with negligible probability, any non-negligible distinguishing advantage for $\mathcal{A}$ in the security experiment against our compiled TLP must arise from executions in which **AllKeysSeq** occurs. Hence, to complete the proof of security, it suffices to argue that any execution of the TLP without setup where **AllKeysSeq** offers only negligible distinguishing advantage to $\mathcal{A}$. The rest of the proof focuses on arguing this. More precisely, if $\epsilon = \mathbf{Adv}_{\text{plain}}(\mathcal{A})$, then

$$\Pr[\text{Good} \wedge \text{AllKeysSeq}] \geq 2\epsilon - \Pr[\text{Bad}].$$

Indeed, $\mathcal{A}$ succeeds with probability exactly $1/2$ on $\text{Good} \wedge \neg \text{AllKeysSeq}$, at most 1 on the other events, and $\Pr[\text{Bad}]$ is negligible.

If $T_1 = 1$, the compiled puzzle consists of a single underlying puzzle. A direct reduction gives the two messages from $\mathcal{A}_1$ to the reusable-security challenger. It then gives the challenge puzzle to $\mathcal{A}_2$ and returns $\mathcal{A}_2$'s output. Its second phase has depth $d = o(T_0)$. We therefore assume that $T_1 \geq 2$ below and set $L = T_1 - 1$.

Consider an execution in which **AllKeysSeq** occurs. Let $\tau_0 = 0$. For $i \in [L]$, let τ_i be the first depth at which $\mathcal{A}_2$ queries $H(k_i)$. Conditioned on **Good**, these depths are strictly increasing. Define $\Delta_i = \tau_i - \tau_{i-1}$. Then

$$\sum_{i=1}^L \Delta_i = \tau_L \leq d.$$

Select $I \leftarrow [L]$ independently of the execution, and set $\theta = \frac{2d}{L}$. For every completed execution, at least half of the indices satisfy $\Delta_I \leq \theta$. Otherwise, the sum of the intervals would exceed d . Also,

$$\theta = \frac{2d}{T_1 - 1} \leq \frac{4d}{T_1} = o(T_0).$$

Lemma 3. *There is a polynomial-size adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the reusable security of the underlying symmetric TLP. Its second phase has depth at most $\lceil \theta \rceil + O(1)$. Moreover,*

$$\mathbf{Adv}_{\text{sym}}(\mathcal{B}) \geq \frac{1}{4} \Pr[\text{Good} \wedge \text{AllKeysSeq}] - \text{negl}(\lambda).$$

Proof. The first phase $\mathcal{B}_1$ performs the following steps.

1. It samples $I \leftarrow [L]$ and runs $\mathcal{A}_1$. It simulates H by lazy sampling. Let $(s_0, s_1, \text{st}_{\mathcal{A}})$ be the output of $\mathcal{A}_1$.
2. It samples a bit $b' \leftarrow \{0, 1\}$ and two candidate keys $\kappa_0, \kappa_1 \leftarrow \{0, 1\}^\lambda$. It sets the unknown chain key to $k_I = \kappa_\beta$, where β will be the challenge bit of the underlying experiment.
3. It samples every other intermediate key. It uses its puzzle-generation oracle to obtain all puzzles except Z_I . In particular, it obtains $Z_{T_1} \leftarrow \text{PGen}(\text{st}, s_{b'})$.
4. It builds the compiled challenge. For known predecessor keys, it uses honest masks. If $I > 1$, it samples $\text{ct}_I \leftarrow \{0, 1\}^{m_z}$ because Z_I is not yet available. It also samples $\text{ct}_{I+1} \leftarrow \{0, 1\}^{m_z}$ because $H(k_I)$ is not yet available.
5. If $I = 1$, it does not start $\mathcal{A}_2$. If $I > 1$, it starts $\mathcal{A}_2$ and runs it until the first query to k_{I-1} . It stops immediately before it answers that query.
6. It gives (κ_0, κ_1) to its challenger and transfers the full simulation state to $\mathcal{B}_2$.

If a required predecessor query does not occur, $\mathcal{B}$ sets an abort flag. It also sets this flag if an intermediate key was queried too early or if relevant sampled strings collide. On an abort, $\mathcal{B}_2$ outputs a uniform bit. These events are covered by **Bad** or have probability $\text{negl}(\lambda)$.

The challenger samples β and gives $Z^* \leftarrow \text{PGen}(\text{st}, \kappa_\beta)$ to $\mathcal{B}_2$. The second phase performs the following steps.

1. If $I = 1$, it installs Z^* as Z_1 and starts $\mathcal{A}_2$. If $I > 1$, it sets $H(k_{I-1}) = \text{ct}_I \oplus Z^*$ and answers the pending query.
2. It runs $\mathcal{A}_2$ for at most $\lceil \theta \rceil + O(1)$ further depth. It stops immediately when $\mathcal{A}_2$ queries κ_0 or κ_1 .
3. It outputs the index of the queried candidate. If both candidates occur in the same query layer, or if neither candidate occurs in time, it outputs a uniformly randomly sampled bit.

The programming in the first step occurs at a fresh point. Thus, the simulation has the real distribution. The uniform value ct_{I+1} is also exact until a candidate is queried. At that query, the reduction stops before it must answer $H(k_I)$.

Fix a good execution in which **AllKeysSeq** occurs. If $\Delta_I \leq \theta$, the reduction reaches the first query to the actual key $k_I = \kappa_\beta$ within its depth budget. A query to the unused candidate is an independent guess and has negligible probability. Since at least half of the indices are short,

$$\text{Adv}_{\text{sym}}(\mathcal{B}) \geq \frac{1}{2} \cdot \frac{1}{2} \Pr[\text{Good} \wedge \text{AllKeysSeq}] - \text{negl}(\lambda).$$

The first factor $1/2$ converts a certain correct guess on the extraction event into advantage over a uniform guess. The second factor is the probability of a short interval. This proves the lemma.

Remark 6. The reduction samples I independently of the execution. Therefore, the short-interval probability is at least $1/2$ for every fixed execution. The bound remains valid after conditioning on $\text{Good} \wedge \text{AllKeysSeq}$.

Remark 7. The algorithm $\mathcal{B}_2$ runs $\mathcal{A}_2$ for at most $\lceil \theta \rceil + O(1)$ depth. It answers random-oracle queries by lazy sampling. Under the round convention in [Section 3.1](#), the fixed local-work allowance covers this simulation. Hence, $\mathcal{B}_2$ has depth $o(T_0)$.

By [Lemmas 2](#) and [3](#),

$$\text{Adv}_{\text{sym}}(\mathcal{B}) \geq \frac{\epsilon}{2} - \frac{1}{4} \Pr[\text{Bad}] - \text{negl}(\lambda).$$

Thus, a non-negligible ROM advantage for $\mathcal{A}$ gives a non-negligible reusable-security advantage for $\mathcal{B}$, which is a contradiction. This completes the proof of security for our compiler.

Analysis in the QROM. We also prove post-quantum security of the above compiler via a QROM analogue of [Theorem 1](#), as stated below.

Theorem 2 (Our Compiler in the QROM). *[Informal Version of [Theorem 6](#)] Under the assumption that $(\text{PSetup}, \text{PGen}, \text{PSolve})$ is a symmetric TLP with setup that satisfies post-quantum symmetric reusable security, and under the assumption that H is a quantum random oracle, the compiler yields a plain TLP without setup that satisfies correctness and post-quantum security in the QROM, and achieves the same efficiency properties as summarized in [Theorem 1](#).*

See [Appendix A](#) for the detailed analysis in the QROM.

Remark 8. The above compiler can obviously also be applied to any (asymmetric) TLP with setup as per [Definition 4](#), where setup and puzzle generation only share some public parameters pp but no secret state st .

Beyond $\sqrt{T}$? Bitansky and Garg [\[10\]](#) showed how to boost the $\sqrt{T}$ succinctness of the lattice-based TLP of [\[3\]](#) to T^ϵ for any arbitrary $0 < \epsilon < 1$. A natural question is whether similar succinctness boosting can be applied to our construction. Unfortunately, at present, we are unable to do so, since their technique relies heavily on sophisticated lattice machinery such as “Laconic Function Evaluation” which is very difficult to realize in the isogeny setting. We leave this as a fascinating open problem.

5 Post-Quantum Symmetric TLP with Setup

In this section, we present our post-quantum symmetric TLP with setup based on the sequentiality of pushforward isogeny computations. Compiling this construction using the compiler in [Section 4.2](#) yields the first post-quantum $\sqrt{T}$ -efficiency TLP without setup/preprocessing from isogeny-based assumptions.

Algorithm 1 Detlsog: seeded deterministic isogeny generation

Input: An elliptic curve E , a degree ℓ , a delay parameter T , a seed $\text{seed}_\phi \in \{0, 1\}^\lambda$

Output: An ℓ^T -isogeny $\phi : E \rightarrow E'$

- 1: Set $E_0 = E$ and $B_0 = \{O\}$
 - 2: **for** $i \in [0, T - 1]$ **do**
 - 3: $K_i \leftarrow H_{\text{walk}}(\text{seed}_\phi, E_i, B_i, i)$
 - 4: Compute the isogeny $\phi_i : E_i \rightarrow E_{i+1}$ with kernel $\ker \phi_i = \langle K_i \rangle$
 - 5: Set $B_{i+1} = \ker \hat{\phi}_i$ $\triangleright$ Excluding B_{i+1} prevents backtracking
 - 6: **return** $\phi = \phi_{T-1} \circ \dots \circ \phi_0$
-

5.1 Construction of Isogeny-based Symmetric TLP with Setup

In our construction, we assume an efficient algorithm that takes as input a security parameter λ and outputs a tuple (p, N, ℓ) and a curve E . The values p, N, ℓ are all primes with $p = \Theta(2^{3\lambda})$, $N = \Theta(2^{2\lambda})$, and $\ell = O(1)$. The curve E is (statistically close to) uniformly distributed among all supersingular elliptic curves over $\mathbb{F}_{p^2}$. The curve can be sampled using standard algorithms [6].

In this construction, we rely on two independent hash functions. The first is H_{walk} , which is used by Detlsog to sample a random isogeny. We define

$$H_{\text{walk}}(\text{seed}_\phi, E_i, B_i, i) \in E_i[\ell] \setminus B_i,$$

where the output is a uniformly sampled point in $E_i[\ell] \setminus B_i$, and excluding the subgroup B_i prevents backtracking. Formally, since $|E_i[\ell] \setminus B_i| = \ell^2 - \ell$, we view H_{walk} as a random oracle with fixed range $[\ell^2 - \ell]$, followed by a canonical bijection, determined by (E_i, B_i) , from this range to $E_i[\ell] \setminus B_i$.

The second hash function that we use is of the form $H_{\text{pf}} : \mathbb{F}_{p^2} \rightarrow \{0, 1\}^\lambda$: for an elliptic curve G over $\mathbb{F}_{p^2}$, write $H_{\text{pf}}(G) = H_{\text{pf}}(j(G))$. Thus, isomorphic curves lead to the same hash value.

Finally, our construction relies on a seeded deterministic algorithm for sampling a long ℓ^T -isogeny. For a fixed seed, the algorithm always generates the same walk; a fresh uniform seed selects a fresh walk. The algorithm is summarized in [Algorithm 1](#). Our TLP construction is as follows:

PSetup $(1^\lambda, T)$: Given a security parameter λ and a sequentiality parameter T , the setup algorithm proceeds as follows:

1. Sample (p, N, ℓ) such that $p = \Theta(2^{3\lambda})$, $N = \Theta(2^{2\lambda})$, $\ell = O(1)$ are primes.
2. Sample a uniformly random supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ by taking a long random walk of length $O(\log p)$ in the ℓ -isogeny graph, starting from a curve E_0 with known endomorphism ring. The setup stores the walk, from which it can derive $\text{End}(E)$.
3. Sample $\text{seed}_\phi \leftarrow \{0, 1\}^\lambda$ and generate an ℓ^T -isogeny $(\phi : E \rightarrow E') \leftarrow \text{Detlsog}(E, \ell, T; \text{seed}_\phi)$ via [Algorithm 1](#).
4. Compute a shortcut isogeny, i.e., a short isogeny ϕ_{short} that acts on the N -torsion of E as the isogeny ϕ , i.e., $\phi_{\text{short}}|_{E[N]} = \phi|_{E[N]}$.

5. Output $\text{st} = (\text{td}, \text{pp})$ where

$$\text{td} = (\text{End}(E), \phi_{\text{short}}), \quad \text{pp} = ((p, N, \ell, T), (E, E'), \text{seed}_\phi)$$

$Z \leftarrow \text{PGen}(\text{st}, s \in \{0, 1\}^\lambda)$: Given a state st and a solution $s \in \{0, 1\}^\lambda$, the puzzle generation algorithm proceeds as follows:

6. Parse st as $\text{st} = ((\text{End}(E), \phi_{\text{short}}), \text{pp})$.
7. Sample a random N -isogeny $\psi : E \rightarrow F$, using knowledge of $\text{End}(E)$.
8. Compute the pushforward $[\phi_{\text{short}}]_* \psi : E' \rightarrow F'$.
9. Compute $\text{ct} = \text{H}_{\text{pf}}(F') \oplus s$.
10. Output the puzzle instance $Z = (\text{pp}, \psi, \text{ct})$.

$s \leftarrow \text{PSolve}(Z)$: Given the puzzle instance $Z = ((p, N, \ell, T), (E, E'), \text{seed}_\phi), \psi, \text{ct})$, do the following:

11. Reconstruct $\phi \leftarrow \text{DetIsog}(E, \ell, T; \text{seed}_\phi)$.
12. Compute the pushforward $[\phi]_* \psi : E' \rightarrow F''$.
13. Compute $s' = \text{H}_{\text{pf}}(F'') \oplus \text{ct}$.
14. Output s' as the solution.

Remark 9. A standard-model construction is also possible using a seeded hardcore predicate, such as the Goldreich–Levin predicate [23], on the encoded j -invariant. Each instance then masks one solution bit, and a construction with λ -bit solutions uses λ independent instances in parallel. The standard-model construction would also require a slightly stronger assumption where the isogeny ϕ , rather than being sampled at random, is sampled from a hash-free variant of DetIsog .

5.2 Finding a Shortcut

In this section, we introduce the main algorithmic building block that is at the core of the TLP construction: an algorithm that allows the puzzle generator, who knows the endomorphism ring trapdoor, to find a shortcut and generate TLPs efficiently (after a long pre-computation.) More precisely, the algorithm receives in input a long isogeny $\phi : E \rightarrow E'$ (and the endomorphism ring $\text{End}(E)$) and computes a short isogeny $\phi_{\text{short}} : E \rightarrow E'$ such that the pushforward of N -isogenies under ϕ and ϕ_{short} is the same. We write $\phi_{\text{short}}|_{E[N]} = \phi|_{E[N]}$ to say that $\phi_{\text{short}}(P) = [\mu]\phi(P)$ for all points $P \in E[N]$ and some fixed $\mu \in \mathbb{Z}_N^\times$. This short isogeny ϕ_{short} lets us compute the puzzle generation efficiently since evaluating ϕ_{short} takes time $O(\text{poly}(\lambda))$, independently of T .

To construct such an algorithm, we first note that two isogenies $\phi, \phi' : E \rightarrow E'$ compute the same pushforward of N -isogenies if and only if their action on the N -torsion is the same, *up to a scalar*: this means that $\phi(P) = [\mu]\phi'(P)$ for any point $P \in E[N]$ and some fixed value $\mu \in \mathbb{Z}_N^\times$. Thus, given a long isogeny $\phi : E \rightarrow E'$, we can first compute the associated ideal I_ϕ by using standard isogeny-to-ideal algorithms [2]; then, we sample a small primitive quaternion α in $I_\phi \cap (\mathbb{Z} + N\mathcal{O})$ where $\mathcal{O}$ is the maximal order associated with $\text{End}(E)$. The quaternion α must

have reduced norm coprime with N ; this happens with overwhelming probability, but if the condition is not satisfied, we resample until it is. The short isogeny ϕ_{short}^1 is then defined as the isogeny associated with the ideal $J = I_\phi \bar{\alpha} / \text{nrm}(I_\phi)$, which can be computed with ideal-to-isogeny algorithms [2]. These computations are summarized in [Algorithm 2](#).

Algorithm 2 Short equivalent isogeny computation

Input: An isogeny $\phi : E \rightarrow E'$, a degree N , an endomorphism ring $\text{End}(E)$

Output: A short isogeny $\phi_{\text{short}} : E \rightarrow E'$ such that $\phi_{\text{short}}(P) = [\mu]\phi(P)$ for $P \in E[N]$, with $\mu \in \mathbb{Z}_N^\times$ and $\gcd(\deg \phi_{\text{short}}, N) = 1$

- 1: Compute the ideal I_ϕ associated with ϕ $\triangleright$ [2, Alg. 3.17]
 - 2: Compute the maximal order $\mathcal{O}$ associated with $\text{End}(E)$
 - 3: Sample a small primitive quaternion $\alpha \in I_\phi \cap (\mathbb{Z} + N\mathcal{O})$ of reduced norm coprime to N
 - 4: Set $J = I_\phi \bar{\alpha} / \text{nrm}(I_\phi)$
 - 5: Compute the isogeny ϕ_{short} associated with J $\triangleright$ [2, Alg. 3.13]
 - 6: **return** ϕ_{short}
-

The isogeny ϕ_{short} preserves the pushforward of N -isogenies: the endomorphism α , since it was chosen in $I_\phi \cap (\mathbb{Z} + N\mathcal{O})$, can be written as $\alpha = \alpha_{\mathbb{Z}} + N\alpha'$ with $\alpha_{\mathbb{Z}} \in \mathbb{Z}$. Thus, for any point $P \in E[N]$, we have

$$\phi_{\text{short}}(P) = \left[\frac{1}{\deg(\phi)} \right] \phi \hat{\alpha}(P) = \left[\frac{1}{\deg(\phi)} \right] \phi(\alpha_{\mathbb{Z}} + N\alpha')(P) = \left[\frac{\alpha_{\mathbb{Z}}}{\deg(\phi)} \right] \phi(P),$$

where $[1/\deg(\phi)]$ denotes multiplication by the inverse of $\deg \phi$ modulo N , which exists since $\gcd(\deg \phi, N) = 1$. Moreover, the isogeny ϕ_{short} is short: its degree is $\deg \phi_{\text{short}} = O(\sqrt{p}N^{3/2})$ [13, Cor. 10], and it is thus independent of T .

Theorem 3 (Correctness of Our Symmetric TLP with Setup). *For any security parameter λ , any sequentiality parameter T , and any solution $s \in \{0, 1\}^\lambda$, letting $\text{st} \leftarrow \text{PSetup}(1^\lambda, T)$, $Z \leftarrow \text{PGen}(\text{st}, s)$, we have $\text{PSolve}(Z) = s$.*

Proof. Note that, by construction, we have $Z = ((p, N, \ell, T), (E, E'), \text{seed}_\phi), \psi, \text{ct}$. The puzzle solver reconstructs $\phi = \text{DetIsog}(E, \ell, T; \text{seed}_\phi)$, computes the curve F'' as the codomain of the pushforward isogeny $[\phi]_*\psi$, and outputs

$$s' = \text{H}_{\text{pf}}(F'') \oplus \text{ct}.$$

The puzzle generation, instead, computes F' as the codomain of the pushforward isogeny $[\phi_{\text{short}}]_*\psi$ and sets $\text{ct} = \text{H}_{\text{pf}}(F') \oplus s$. However, ϕ and ϕ_{short} act on the N -torsion in the same way up to a scalar $\mu \in \mathbb{Z}_N^\times$, and since ψ is a N -isogeny,

$$\phi_{\text{short}}(\ker \psi) = \langle [\mu]\phi(K_\psi) \rangle = \langle \phi(K_\psi) \rangle = \phi(\ker \psi).$$

¹ The isogeny ϕ_{short} is defined up to post-composition with an isomorphism. To obtain the correct isogeny ϕ_{short} with codomain E' , it may be necessary to compose the output of the ideal-to-isogeny translation with an isomorphism to E' .

Here, K_ψ generates $\ker \psi$. The two pushforwards have the same kernel, so their codomains are isomorphic and $j(F') = j(F'')$. Hence, we have

$$s' = H_{\text{pf}}(F'') \oplus \text{ct} = H_{\text{pf}}(F') \oplus \text{ct} = s,$$

which concludes the proof of correctness.

5.3 Parameters and Efficiency

Parameters. Let λ be the security parameter and T be the sequentiality parameter. Note again that in general, the sequentiality parameter T is a (large) polynomial in λ . For the TLP construction above, we set $p = \Theta(2^{3\lambda})$, $N = \Theta(2^{2\lambda})$, and $\ell = O(1)$. Observe that all of the above parameters are independent of the sequentiality parameter T , which only determines the length of the non-backtracking walk in the ℓ -isogeny $\mathbb{F}_{p^2}$ -graph taken by the setup algorithm and the puzzle solver.

Remark 10 (Choice of N). The construction and conservative parameters presented here ensure correctness and asymptotic efficiency. Concrete implementation may opt for more efficient choices of parameters: for instance, choosing the prime p so that $(p+1)$ is divisible by a large power of 2 makes it easier to work with HD representations. Similarly, choosing N of the form $q(2^e - q)$, for some q , leads to a 2D representation, which is practically faster to work with. Accordingly, in our concrete implementation in [Section 6](#), we use the implementation-oriented choice $N = q(2^e - q)$, which is composite and therefore falls outside the prime- N asymptotic formulation above, but enables a substantially more efficient two-dimensional representation.

Efficiency Analysis. We now analyze the efficiency of setup, puzzle generation, and puzzle solving.

Efficiency of Setup. First we argue that **PSetup** takes time $O(T \cdot \text{poly}(\lambda))$ for the choice of parameters described above. To see this, observe that the setup algorithm starts by sampling a uniformly random supersingular elliptic curve: this can be done efficiently, up to a negligible statistical distance, by taking a walk of length $O(\log p)$ in the ℓ -isogeny graph [6]; the setup translates the walk into an ideal to efficiently obtain $\text{End}(E)$, which provides the trapdoor component of **st**. Then, the setup algorithm computes a non-backtracking walk of length T in the ℓ -isogeny $\mathbb{F}_{p^2}$ -graph using **DetIsog** ([Algorithm 1](#)), which takes time $O(T \cdot \text{poly}(\lambda))$. The public parameters store only the λ -bit seed seed_ϕ , rather than the $O(T \cdot \text{poly}(\lambda))$ -bit factorization of ϕ . The setup also computes a shortcut ϕ_{short} , which takes time $O(T \cdot \text{poly}(\lambda))$, as argued in [Section 5.2](#). Moreover, since $p = \Theta(2^{3\lambda})$, all of the other computations performed by the setup take time $O(\text{poly}(\lambda))$. Hence, overall, setup takes time $O(T \cdot \text{poly}(\lambda))$.

Efficiency of Puzzle Generation. Next, we argue that PGen takes time $\text{poly}(\lambda)$. Puzzle generation first samples an isogeny of degree $N = \Theta(2^{2\lambda})$ from E by relying on the knowledge of $\text{End}(E)$. This can be done efficiently in time $O(\text{poly}(\lambda))$ [5, Theorem 1]. Then, PGen computes the pushforward $[\phi_{\text{short}}]_*\psi$, where both isogenies have degree independent of T . This is also efficient [32, Proposition 6.15]. Computing $j(F')$ takes polynomial time in λ ; masking s then takes one query to H_{pf} and $O(\lambda)$ bit operations. Thus, the entire puzzle generation procedure takes time $O(\text{poly}(\lambda))$, independently of T .

Efficiency of Puzzle Solving. Finally, we argue that puzzle solving can be done in time $O(T \cdot \text{poly}(\lambda))$ for the choice of parameters described above. The solver reconstructs the walk $\phi = \text{DetIsog}(E, \ell, T; \text{seed}_\phi)$ from the public seed and computes the pushforward $[\phi]_*\psi$ along that walk. This takes time $O(T \cdot \text{poly}(\lambda))$ (this also follows from [32, Proposition 6.15]). Recovering s then requires one query to H_{pf} on $j(F'')$ and an XOR, so the overall running time remains $O(T \cdot \text{poly}(\lambda))$.

5.4 Sequentiality Assumption and Security

At the core of our symmetric TLP with setup is the computation of isogeny pushforwards. Given two isogenies $\phi : E \rightarrow E_A, \psi : E \rightarrow E_B$ with the same domain E , the pushforward of ψ under ϕ , denoted by $[\phi]_*\psi$, is an isogeny from E_A whose kernel is $\ker([\phi]_*\psi) = \phi(\ker \psi)$. Alternatively, if K generates the kernel of ψ , the kernel of its pushforward under ϕ is generated by $\phi(K)$.

Fundamentally, the symmetric reusable security of our construction relies on the sequentiality of pushforward computations: given a random cyclic ℓ^T -isogeny $\phi_T : E \rightarrow E'$ and an isogeny $\psi : E \rightarrow F$ of large prime degree N , given in HD representation, computing the codomain of the pushforward $[\phi_T]_*\psi$ must take time at least T . This should hold even when the adversary has seen ϕ and is allowed pre-computations unbounded in T . More formally, we have the following.

Assumption 1 (Sequentiality of Pushforward Isogeny Computations). Given a security parameter λ and a sequentiality parameter $T = \text{poly}(\lambda)$: let $p = \Theta(2^{3\lambda})$ and $N = \Theta(2^{2\lambda})$ be primes, and let $\ell = O(1)$ be a small prime. Let $E/\mathbb{F}_{p^2}$ be a supersingular elliptic curve with unknown endomorphism ring. Consider the following game between a challenger and a polynomial-sized (quantum) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where the first phase $\mathcal{A}_1$ runs in time $\text{poly}(\lambda, T)$, while the second phase $\mathcal{A}_2$ runs in parallel time $o(T)$:

1. $\mathcal{A}_1$ receives as input $(p, N, \ell, E, E', \phi, \{\psi_i\}_{i \in [q]})$ from the challenger, where $\phi : E \rightarrow E'$ is a random cyclic ℓ^T -isogeny, sampled by a uniform non-backtracking walk of length T and represented by its degree- ℓ factors, and each $\psi_i : E \rightarrow F_i$ is a uniformly sampled isogeny of degree N , given in HD representation.
2. $\mathcal{A}_1$ eventually outputs a state $\text{st}_{\mathcal{A}}$.
3. The challenger uniformly randomly samples $\psi : E \rightarrow F$ among the N -isogenies from E , and provides $\mathcal{A}_2$ with $(\text{st}_{\mathcal{A}}, \psi)$, where ψ is again given in HD representation.

4. $\mathcal{A}_2$ eventually outputs a curve $F'_\mathcal{A}$.

Write F' for the codomain of the pushforward isogeny $[\phi]_*\psi : E' \rightarrow F'$. Then for any λ, T and any probabilistic polynomial-sized adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ runs in time $\text{poly}(\lambda, T)$ and $\mathcal{A}_2$ runs in parallel time $o(T)$, we have

$$\Pr[j(F') = j(F'_\mathcal{A})] \leq \text{negl}(\lambda).$$

We allow the adversary $\mathcal{A}$ a pre-computation taking total time $\text{poly}(\lambda, T)$. Intuitively, this makes the assumption *multi-use*, in the sense that we want it to be computationally intractable for $\mathcal{A}$ to evaluate $[\phi]_*\psi$ for uniformly random $\psi \leftarrow \text{isog}_N(E)$, even given an unbounded amount of parallel resources, *and even if* $\mathcal{A}$ is allowed $\text{poly}(\lambda)$ pre-computation after $(E, E', N, \phi, \{\psi_i\})$ are generated.

Justification of Assumption 1. In this subsection, we present an expanded justification of our conjecture that Assumption 1 is valid. We first argue why *computing one pushforward* is an inherently sequential computation, and then we show why having access to multiple isogenies and their pushforwards does not make the problem easier.

Given an ℓ^T -isogeny $\phi : E \rightarrow E'$ and a N -isogeny $\psi : E \rightarrow F$, there are several methods to compute the pushforward $\psi' = [\phi]_*(\psi)$:

1. Evaluating $K' = \phi(K_\psi)$, where K_ψ generates the kernel of ψ , and computing the isogeny ψ' with kernel $\langle K' \rangle$. An adversary could try to precompute the action of ϕ on an N -torsion basis. Even granting efficient access to this basis and to K' , the remaining task is to compute the quotient curve $E'/\langle K' \rangle$. For prime $N = \Theta(2^{2\lambda})$, the square-root Vélú algorithm computes this codomain in $\tilde{O}(\sqrt{N}) = \tilde{O}(2^\lambda)$ operations in a field containing the kernel generator [9]. The cost of this direct kernel-based method motivates our conservative choice of N , even when the kernel is already available. Security against arbitrary polynomial-sized quantum adversaries remains the conjecture stated in Assumption 1.
2. Computing $[\psi]_*\phi$. To do this, the adversary would write the isogeny ϕ as the composition $\phi = \phi_T \circ \dots \circ \phi_1$ of T isogenies of degree ℓ .¹ Then, they would compute $\psi(\ker(\phi_1))$, use this to obtain $[\phi_1]_*\psi$, and repeat the process to push all components of ϕ . Such a computation has been called an SIDH ladder in the literature (see [6] for more details). This approach requires computing the isogeny $[\psi]_*\phi$, which is an ℓ^T -isogeny, and is thus expected to take parallel time $O(T)$.
3. It is also possible to efficiently compute pushforwards by using higher-dimensional isogenies [32, Proposition 6.15] However, the higher-dimensional isogeny used in this computation is an isogeny of smooth degree greater than $2^T + N$, which in turn means that it would take $\log(2^T + N) > T$

¹ If the torsion of order ℓ^t , for some $1 < t < T$, is easily accessible, the same approach works with decomposing ϕ into fewer isogenies, but this does not affect the runtime dependency on T .

sequential steps. Note that assuming that higher-dimensional computations are sequential is not new: since one-dimensional isogenies can be embedded into higher-dimensional ones, a parallel computation of HD isogenies would break existing isogeny-based constructions such as [20,14].

4. Other methods may include computing the ideals corresponding to ϕ and ψ to do the pushforward computation on the ideal side. Alternatively, the adversary may hope to find a shorter isogeny from E to E' , which would allow them to compute pushforwards in parallel time regardless of T . However, both of these approaches require solving the endomorphism ring problem, which is assumed to be hard because $\log p > 2\lambda$.

Now, consider the case of an adversary that uses its $\text{poly}(\lambda, T)$ pre-computation to speed up the pushforward computation. First, note that the adversary cannot find a “shortcut”, an isogeny ϕ_{short} between E and E' that is shorter than ϕ : if they did, the isogeny $\hat{\phi} \circ \phi_{\text{short}}$ would be a non-trivial endomorphism on E , and the adversary could be turned into a generic solver for the Endomorphism Ring problem [29].

The only meaningful pre-computation that an adversary could do is compute the pushforwards of all the ψ_i isogenies, hoping that having q pushforwards of N -isogenies under ϕ would speed up the computation of $[\phi]_*\psi$ for a new N -isogeny ψ . Clearly, if $\psi = \psi_i$ for some i , the adversary would trivially break the assumption: however, there are $N + 1 = \Theta(2^{2\lambda})$ isogenies of degree N , and thus the probability that $\psi = \psi_i$ is negligible.

To see why having precomputed multiple pushforwards does not help with the computation of a new one, consider that in the computation of $[\phi]_*\psi$, the isogeny ψ acts as a randomizer. Regardless if the pushforward is computed via an SIDH ladder (Method 2) or via HD isogenies (Method 3), the sequential T -step isogeny (either in dimension 1 or in higher dimension) changes every time.

Security in the ROM. We state the following theorem, which captures the security of our symmetric TLP with setup construction in the ROM.

Theorem 4 (ROM Security of Our Symmetric TLP with Setup). *Under Assumption 1, the symmetric TLP with setup above achieves reusable security against classical adversaries in the ROM.*

Proof. We model the two hash functions H_{walk} and H_{pf} as independent random oracles. Suppose that a polynomial-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the symmetric TLP security has advantage ϵ . Its first phase makes at most q_{PG} classical puzzle-generation queries. Let q_H bound its H_{pf} queries across both phases. Its second phase has depth $o(T)$. We construct an adversary $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ for Assumption 1. It uses $q = q_{\text{PG}}$ prechallenge isogenies.

Embedding the random walk. Before running $\mathcal{A}_1$, $\mathcal{D}_1$ samples $\text{seed}_\phi \leftarrow \{0, 1\}^\lambda$ and programs the T inputs to H_{walk} encountered by $\text{DetIsog}(E, \ell, T; \text{seed}_\phi)$ to uniformly random generators of the kernels of the supplied factors ϕ_i . Thus $\text{DetIsog}(E, \ell, T; \text{seed}_\phi) = \phi$. These inputs are distinct, and the adversary has made no oracle queries when they are programmed. Since each supplied step and each

programmed generator are uniform among the admissible choices, the resulting joint distribution of $(\text{seed}_\phi, \phi, H_{\text{walk}})$ is exactly the real one. Both reduction phases answer every other entry of both random oracles by lazy sampling. Let L record every evaluated H_{pf} input, including the evaluations made while answering puzzle-generation queries.

Simulating the puzzles. The algorithm $\mathcal{D}_1$ runs $\mathcal{A}_1$ and simulates its puzzle-generation oracle $\mathcal{O} = \text{PGen}(\text{st}, \cdot)$. On the i -th query $s \in \{0, 1\}^\lambda$, it assigns the supplied isogeny ψ_i to that query. The isogenies ψ_i are independent and uniform, so they have exactly the distribution of the fresh randomness used by the real puzzle-generation oracle, even when the queries are adaptive. The reduction does not know $\text{End}(E)$ or ϕ_{short} , but it can use ϕ to compute the codomain F'_i of $[\phi]_*\psi_i$ (at this stage, the reduction is not bounded by T). This is the same codomain as that of $[\phi_{\text{short}}]_*\psi_i$, which the real puzzle generator computes. The reduction evaluates H_{pf} at F'_i and returns

$$Z_i = (\text{pp}, \psi_i, H_{\text{pf}}(F'_i) \oplus s),$$

where $\text{pp} = ((p, N, \ell, T), (E, E'), \text{seed}_\phi)$. Notice that the supplied factorization of ϕ is used only by the reduction; the simulated public parameters contain the same succinct seeded description as the real construction. Thus, the simulated oracle answers, including the random oracle answers, have the same joint distribution as in the real first phase. Computing all of them costs $q_{\text{PG}}T \text{poly}(\lambda)$, which is within $\mathcal{D}_1$'s time limit for the first phase. When $\mathcal{A}_1$ outputs candidate solutions $s_0, s_1 \in \{0, 1\}^\lambda$ and state $\mathbf{z}$, the state passed from $\mathcal{D}_1$ to $\mathcal{D}_2$ contains $(s_0, s_1, \mathbf{z}, \text{pp})$. The fresh isogeny ψ supplied to $\mathcal{D}_2$ by the assumption is independent and uniform, just as in the challenge call to PGen . To keep its depth $o(T)$, $\mathcal{D}_2$ does not compute the fresh pushforward; instead, it samples $b \leftarrow \{0, 1\}$ and an independent mask $u \leftarrow \{0, 1\}^\lambda$, and forms

$$Z = (\text{pp}, \psi, u \oplus s_b),$$

which defines the simulated game G_{sim} . It runs $b' \leftarrow \mathcal{A}_2(Z, \mathbf{z})$ once. After $\mathcal{A}_2$ terminates, $\mathcal{D}_2$ outputs a curve isomorphism class whose j -invariant is an entry of L selected uniformly at random, or a fixed curve if L is empty, where L is the list of H_{pf} query inputs in the simulation.

To analyze its success, let F' be the codomain of the fresh challenge pushforward $[\phi]_*\psi : E' \rightarrow F'$ and set $x^* = j(F')$. In G_{sim} , the independent mask u hides b perfectly, so $\Pr_{G_{\text{sim}}}[b' = b] = 1/2$. For comparison, define G_{real} whose oracle simulator also knows x^* . It programs $H_{\text{pf}}(x^*) = u$ at challenge generation if that input is fresh; otherwise it uses the existing oracle value as the mask. This has the real-game distribution. The views can differ only if x^* is queried, including before the challenge. Hence $\epsilon \leq \Pr_{G_{\text{sim}}}[x^* \in L]$.

Success and depth. Since $|L| \leq Q := q_H + q_{\text{PG}}$, the success probability of $\mathcal{D}_2$ is at least ϵ/Q for $Q > 0$; if $Q = 0$, then $\epsilon = 0$. Under our round convention, $\mathcal{D}_2$ has depth $O(\text{depth}(\mathcal{A}_2) + 1) = o(T)$ and polynomial size, contradicting [Assumption 1](#).

Post-Quantum Security in the QROM. We also prove post-quantum security of our symmetric TLP with setup construction against quantum adversaries.

Table 1. Puzzle-generation costs using four parallel PGen workers on an Apple M3 CPU. Setup and solving remain single-core. Non-italicized values are measured; italicized values are extrapolated from measurements at shorter delays. Online puzzle generation excludes setup, while total generation includes it.

Target Solve Time	T_0	T_1	Setup	Online Puzzle Gen	Total Puzzle Gen	Puzzle Size
1 min	134	18	7.7 s	6.7 s	14.4 s	73.5 KB
10 min	268	53	18.6 s	19.8 s	38.5 s	216.5 KB
30 min	402	119	29.5 s	44.6 s	74.1 s	486.1 KB
1 h	871	114	57.8 s	42.7 s	100.5 s	465.7 KB
2 h	1,072	184	68.3 s	68.9 s	2.3 min	751.6 KB
12 h	2,814	412	2.8 min	2.6 min	5.4 min	1.69 MB
1 d	3,886	584	3.9 min	3.6 min	7.6 min	2.39 MB
1 wk	10,050	1,576	<i>10.1 min</i>	<i>9.8 min</i>	<i>19.9 min</i>	6.44 MB
2 wk	14,070	2,249	<i>14.1 min</i>	<i>14.0 min</i>	<i>28.2 min</i>	9.19 MB
1 mo	20,703	3,272	<i>20.8 min</i>	<i>20.4 min</i>	<i>41.2 min</i>	13.37 MB
6 mo	50,384	8,058	<i>50.5 min</i>	<i>50.3 min</i>	<i>100.8 min</i>	32.91 MB
1 yr	71,690	11,481	<i>71.8 min</i>	<i>71.7 min</i>	<i>2.4 h</i>	46.90 MB

Theorem 5 (QROM Security of Our Symmetric TLP with Setup).

Under *Assumption 1*, the symmetric TLP with setup above achieves reusable security in the QROM.

See [Appendix A.7](#) for the detailed proof.

6 Proof-of-Concept Implementation and Benchmarks

In this section, we present a proof-of-concept implementation of the complete TLP. We combine the symmetric TLP with setup of [Section 5](#) with the compiler of [Section 4.2](#), obtaining a plain TLP with an asymptotic quadratic gap between puzzle generation and puzzle solving. To our knowledge, this is the first implementation of a post-quantum TLP in the literature.

Our evaluation has three goals: (i) validate the symmetric TLP and compiler end-to-end; (ii) demonstrate that the resulting plain TLP uses practically implementable algorithms; and (iii) quantify the concrete usefulness of the quadratic generation-solving gap. The implementation is written in SageMath with extensive assistance from OpenAI’s Codex 5.6. The source code is available at <https://anonymous.4open.science/r/pqTLP>. We emphasize that this is a proof of concept; production deployment would require careful parameter tuning and benchmarking of the sequential computation on powerful or dedicated hardware, which is beyond our scope.

Concrete Parameters. Our parameter choices are motivated by SQIsign level-I security with $\lambda = 128$; the implementation readily extends to higher security levels. We choose $p = 3 \cdot 2^{324} - 1$, following the level-I parameters and concrete

Table 2. Puzzle-generation costs for different target delays on an Apple M3 CPU. Non-italic values are measured; italicized values are extrapolated from measurements at shorter delays. Online puzzle generation excludes setup, while total generation includes it.

Target Solve Time	T_0	T_1	Setup	Online Puzzle Gen	Total Puzzle Gen	Puzzle Size
1 min	201	9	13.2 s	11.1 s	24.3 s	36.8 KB
10 min	603	28	43.8 s	34.5 s	78.3 s	114.4 KB
30 min	1,072	46	68.3 s	56.7 s	2.1 min	187.9 KB
1 h	1,340	74	83.5 s	91.2 s	2.9 min	302.3 KB
2 h	1,943	102	117.7 s	2.1 min	4.1 min	416.7 KB
12 h	4,757	239	<i>4.8 min</i>	<i>4.9 min</i>	<i>9.7 min</i>	976.3 KB
1 d	6,767	335	<i>6.8 min</i>	<i>6.9 min</i>	<i>13.7 min</i>	1.37 MB
1 wk	18,090	874	<i>18.1 min</i>	<i>17.9 min</i>	<i>36.1 min</i>	3.57 MB
2 wk	25,527	1,238	<i>25.6 min</i>	<i>25.4 min</i>	<i>51.0 min</i>	5.05 MB
1 mo	37,252	1,817	<i>37.3 min</i>	<i>37.3 min</i>	<i>74.6 min</i>	7.42 MB
6 mo	91,388	4,441	<i>91.5 min</i>	<i>91.2 min</i>	<i>3.0 h</i>	18.14 MB
1 yr	129,913	6,334	<i>2.2 h</i>	<i>2.2 h</i>	<i>4.3 h</i>	25.88 MB

security analysis of the round-3 SQIsign submission [1], including the recent endomorphism-ring attack [35]. We set $N = q(2^{257} - q)$, where q is the smallest prime larger than 2^{256} . Although composite, N contains a 257-bit prime factor, providing the required security against directly computing the degree- N isogeny. Crucially, our choices of p, N permit an efficient two-dimensional representation of ψ using rational torsion points, avoiding generic four- or eight-dimensional representations. Finally, we set $\ell = 2$, since $2^{324} \mid p+1$, and accelerate [Algorithm 1](#) by processing the walk in degree- 2^{67} chunks.¹ For each chunk, we sample a 2^{67} -torsion basis (R_i, S_i) and set $K_i = R_i + [r_i]S_i$, deriving r_i via SHAKE and incrementing a counter in its input whenever K_i backtracks.

Benchmarking Results. We benchmark the prototype on an Apple M3 CPU, choosing T_0 and T_1 for each target delay to minimize total puzzle-generation time, including setup. [Table 1](#) reports results when the independently generated puzzles are parallelized across four threads; setup and solving remain single-core. [Table 2](#) reports serial-generation costs. Parallelization with four cores roughly halves total generation time, but shifts the optimal balance toward smaller T_0 and larger T_1 , reducing setup time at the cost of larger puzzles. Concretely, a puzzle with a one-hour target delay takes 2.9 minutes to generate serially and 100.5 seconds with four threads, including setup. For a one-year delay, the corresponding projections are 4.3 hours and 2.4 hours, with puzzle sizes of 25.88

¹ The vertical isogenies of degree $N = q(2^{257} - q)$ use torsion points of order 2^{257} for their representation. Since $2^{324} \mid p+1$ and $324 - 257 = 67$, splitting the horizontal walk into isogenies of degree 2^{67} allows their two-dimensional representation to be mapped through each chunk without changing the representation dimension.

MB and 46.90 MB. These results demonstrate a meaningful concrete gap between puzzle generation and puzzle solving.

Acknowledgements

We used ChatGPT 5.6 Sol in developing certain technical details of the QROM-based analysis of our compiler, described in [Appendix A](#). We also used OpenAI’s Codex 5.6 for assistance with the implementation in [Section 6](#). All resulting arguments, technical claims, implementation outputs, and experimental results were independently checked and validated by the authors before inclusion in the paper. Finally, all material outside these sections was developed and written solely by the authors without the use of AI.

The first author is supported by the Arvind Raghunathan center for the Theory of Computing (aRtCS) at IIT Madras, the Mphasis F1 Foundation and the National Quantum Mission initiative of Dept of Science & Technology, India.

References

1. Aardal, M.A., Adj, G., Aranha, D.F., Basso, A., Borin, G., Canales Martínez, I.A., Chávez-Saab, J., Corte-Real Santos, M., Dartois, P., De Feo, L., Duparc, M., Espitau, T., Eriksen, J.K., Fouotsa, T.B., Gazzoni Filho, D.L., Hess, B., Invernizzi, R., Kohel, D., Leroux, A., Longa, P., Maino, L., Meyer, M., Mula, M., Nakagawa, K., Onuki, H., Panny, L., Patranabis, S., Petit, C., Pope, G., Reijnders, K., Robert, D., Rodríguez-Henríquez, F., Schaeffler, S., Vercauteren, F., Wallet, A., Wesolowski, B., van Woerden, W.: SQIsign. Tech. rep., National Institute of Standards and Technology (2026), available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-3-additional-signatures>
2. Aardal, M.A., Adj, G., Aranha, D.F., Basso, A., Canales Martínez, I.A., Chávez-Saab, J., Santos, M.C., Dartois, P., De Feo, L., Duparc, M., Eriksen, J.K., Fouotsa, T.B., Filho, D.L.G., Hess, B., Kohel, D., Leroux, A., Longa, P., Maino, L., Meyer, M., Nakagawa, K., Onuki, H., Panny, L., Patranabis, S., Petit, C., Pope, G., Reijnders, K., Robert, D., Rodríguez Henríquez, F., Schaeffler, S., Wesolowski, B.: SQIsign. Tech. rep., National Institute of Standards and Technology (2024), available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures>
3. Agrawal, S., Malavolta, G., Zhang, T.: Time-lock puzzles from lattices. In: Annual International Cryptology Conference. pp. 425–456. Springer (2024)
4. Ambainis, A., Hamburg, M., Unruh, D.: Quantum security proofs using semi-classical oracles. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 269–295. Springer, Cham (Aug 2019). doi: 10.1007/978-3-030-26951-7_10
5. Basso, A., Borin, G., Castryck, W., Corte-Real Santos, M., Invernizzi, R., Leroux, A., Maino, L., Vercauteren, F., Wesolowski, B.: PRISM: Simple and compact identification and signatures from large prime degree isogenies. In: Jager, T., Pan, J. (eds.) PKC 2025, Part III. LNCS, vol. 15676, pp. 300–332. Springer, Cham (May 2025). doi: 10.1007/978-3-031-91826-1_10
6. Basso, A., Codogni, G., Connolly, D., De Feo, L., Fouotsa, T.B., Lido, G.M., Morrison, T., Panny, L., Patranabis, S., Wesolowski, B.: Supersingular curves you can trust. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 405–437. Springer, Cham (Apr 2023). doi: 10.1007/978-3-031-30617-4_14
7. Basso, A., Fouotsa, T.B.: New SIDH countermeasures for a more efficient key exchange. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part VIII. LNCS, vol. 14445, pp. 208–233. Springer, Singapore (Dec 2023). doi: 10.1007/978-981-99-8742-9_7
8. Basso, A., Maino, L.: POKÉ: A compact and efficient PKE from higher-dimensional isogenies. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part II. LNCS, vol. 15602, pp. 94–123. Springer, Cham (May 2025). doi: 10.1007/978-3-031-91124-8_4
9. Bernstein, D.J., De Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. Cryptology ePrint Archive, Report 2020/341 (2020), <https://eprint.iacr.org/2020/341>
10. Bitansky, N., Garg, R.: Succinct randomized encodings from laconic function evaluation, faster and simpler. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 406–436. Springer (2025)
11. Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., Waters, B.: Time-lock puzzles from randomized encodings. In: Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science. pp. 345–356 (2016)

12. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Berlin, Heidelberg (Dec 2011). doi: 10.1007/978-3-642-25385-0_3
13. Borin, G., De Feo, L., Lido, G., Schaeffler, S.: The sqinstructor: a guide to sqsign and the deuring correspondence with level structures. In: Heninger, N., Rosulek, M. (eds.) Advances in Cryptology – CRYPTO 2026. pp. 537–569. Springer Nature Switzerland, Cham (2026)
14. Burdges, J., De Feo, L.: Delay encryption. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 302–326. Springer, Cham (Oct 2021). doi: 10.1007/978-3-030-77870-5_11
15. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 423–447. Springer, Cham (Apr 2023). doi: 10.1007/978-3-031-30589-4_15
16. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 395–427. Springer, Cham (Dec 2018). doi: 10.1007/978-3-030-03332-3_15
17. Chavez-Saab, J., Rodríguez-Henríquez, F., Tibouchi, M.: Verifiable isogeny walks: towards an isogeny-based postquantum vdf. In: International Conference on Selected Areas in Cryptography. pp. 441–460. Springer (2021)
18. Chung, K.M., Fehr, S., Huang, Y.H., Liao, T.N.: On the compressed-oracle technique, and post-quantum security of proofs of sequential work. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 598–629. Springer, Cham (Oct 2021). doi: 10.1007/978-3-030-77886-6_21
19. Dartois, P., Eriksen, J.K., Fouotsa, T.B., Merdy, A.H.L., Invernizzi, R., Robert, D., Rueger, R., Vercauteren, F., Wesolowski, B.: PEGASIS: Practical effective class group action using 4-dimensional isogenies. Cryptology ePrint Archive, Paper 2025/401 (2025), <https://eprint.iacr.org/2025/401>
20. De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 248–277. Springer, Cham (Dec 2019). doi: 10.1007/978-3-030-34578-5_10
21. Decru, T., Maino, L., Sanso, A.: Towards a quantum-resistant weak verifiable delay function. In: International Conference on Cryptology and Information Security in Latin America. pp. 149–168. Springer (2023)
22. Fouotsa, T.B., Moriya, T., Petit, C.: M-SIDH and MD-SIDH: Countering SIDH attacks by masking information. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 282–309. Springer, Cham (Apr 2023). doi: 10.1007/978-3-031-30589-4_10
23. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: 21st ACM STOC. pp. 25–32. ACM Press (May 1989). doi: 10.1145/73007.73010
24. Grilo, A.B., Hövelmanns, K., Hülsing, A., Majenz, C.: Tight adaptive reprogramming in the QROM. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part I. LNCS, vol. 13090, pp. 637–667. Springer, Cham (Dec 2021). doi: 10.1007/978-3-030-92062-3_22
25. Hsieh, Y.C., Lin, H., Luo, J.: Attribute-based encryption for circuits of unbounded depth from lattices: garbled circuits of optimal size, laconic functional evaluation, and more. SIAM Journal on Computing pp. FOCS23–90 (2025)

26. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A direct key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 448–471. Springer, Cham (Apr 2023). doi: 10.1007/978-3-031-30589-4_16
27. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Annual International Cryptology Conference. pp. 620–649. Springer (2019)
28. Page, A., Robert, D.: Introducing clapoti(s): Evaluating the isogeny class group action in polynomial time. Cryptology ePrint Archive, Paper 2023/1766 (2023), <https://eprint.iacr.org/2023/1766>
29. Page, A., Wesolowski, B.: The supersingular endomorphism ring and one endomorphism problems are equivalent. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VI. LNCS, vol. 14656, pp. 388–417. Springer, Cham (May 2024). doi: 10.1007/978-3-031-58751-1_14
30. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
31. Robert, D.: Breaking SIDH in polynomial time. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 472–503. Springer, Cham (Apr 2023). doi: 10.1007/978-3-031-30589-4_17
32. Robert, D.: On the efficient representation of isogenies - A survey for nutmic 2024. In: Dabrowski, A., Pieprzyk, J., Pomykala, J. (eds.) Number-Theoretic Methods in Cryptology - 4th International Conference, NuTMiC 2024, Szczecin, Poland, June 24-26, 2024, Revised Selected Papers. Lecture Notes in Computer Science, vol. 14966, pp. 3–84. Springer (2024). doi: 10.1007/978-3-031-82380-0_1, https://doi.org/10.1007/978-3-031-82380-0_1
33. Rueger, R.: Computing the Isogeny Class-Group Action on Ordinary Elliptic Curves by going into Higher Dimensions. Master thesis, ETH Zurich, Zurich (2024). doi: 10.3929/ethz-b-000700125
34. Unruh, D.: Revocable quantum timed-release encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 129–146. Springer, Berlin, Heidelberg (May 2014). doi: 10.1007/978-3-642-55220-5_8
35. Wesolowski, B.: The supersingular isogeny problem in time and memory $p^{1/3+o(1)}$. Cryptology ePrint Archive, Report 2026/1486 (2026), <https://eprint.iacr.org/2026/1486>
36. Zhandry, M.: How to construct quantum random functions. In: 53rd FOCS. pp. 679–687. IEEE Computer Society Press (Oct 2012). doi: 10.1109/FOCS.2012.37

A Post-Quantum Security of the Compiler in the QROM

In this section, we analyze the post-quantum security of the compiler of [Section 4.2](#) in the *quantum* random oracle model (QROM) [12]. Concretely, the hash function H is modeled as a quantum random oracle, such that a quantum adversary that runs H on its own hardware can evaluate it in superposition.

We define the QROM model, state the corresponding post-quantum analogues of the definitions used in [Theorem 1](#), and finally prove a QROM analogue of [Theorem 1](#). The proof applies to every polynomially bounded T_1 . This includes the balanced choice $T_0 = T_1 = \sqrt{T}$.

Differences from the Classical ROM Analysis. The classical ROM-based analysis in [Theorem 1](#) is built on three transcript-based steps, none of which survives quantum queries as stated:

- (a) the events **Bad** and **AllKeysSeq**, and the times τ_i , are defined in terms of *which points the adversary queries and when*; a quantum query is a superposition $\sum_x \alpha_x |x\rangle$, so there is no query transcript, and measuring queries to create one disturbs the adversary;
- (b) the reduction $\mathcal{B}_1$ *detects* the first query $H(k_{i^*-1})$ in order to decide when to hand over to $\mathcal{B}_2$;
- (c) $\mathcal{B}$ *programs* the oracle adaptively at k_{i^*-1} , and $\mathcal{B}_2$ *detects* queries at κ_0, κ_1 to extract its answer.

Our Approach. We replace these transcript operations with a fixed delayed-activation schedule. The schedule divides the second phase into short deterministic blocks. Each pair of adjacent games differs at one oracle point during a fixed prefix. One Cauchy-Schwarz step combines their query masses. A reduction then embeds one reusable challenge and measures a query in one randomly selected block. This method does not incur a square-root loss at each link. We use fresh adaptive reprogramming only at the endpoint, after the first phase ends. The tight QROM bound for adaptive reprogramming covers this change [24].

We remark here that, in the applications of [18] (hash chains, proofs of sequential work, etc.) the random oracle itself is the sequential resource, and the entire analysis can be carried out inside their query-complexity framework. In our compiler the sequential resource is the underlying symmetric TLP: the compiler oracle H only *glues* the chain, while each link $Z_i \mapsto k_i$ is crossed by running **PSolve**. This cost is not captured by the query complexity of H . The underlying TLP may use its own quantum random oracles, which remain unchanged throughout the compiler proof. Consequently no pure query-complexity argument can prove [Theorem 6](#); the information-theoretic QROM lemmas below must be interleaved with a reduction to the post-quantum security of the underlying TLP.

A.1 Technical Overview of the QROM Extension

We begin with a technical overview of the extension of [Theorem 1](#) to the QROM. The main difficulty is that a quantum execution provides no classical transcript

identifying the first query to each k_i : measuring individual queries would disturb the adversary's state, while detecting a query before programming the oracle is likewise unavailable. We overcome this obstacle using a fixed *delayed-activation schedule*, which allows us to reason about the adversary's progress without observing its individual queries. The underlying symmetric TLP may itself use a tuple $\mathbf{H}_u$ of quantum random oracles, which we take to be independent of the compiler oracle H . Our reduction forwards $\mathbf{H}_u$ unchanged and applies the delayed-activation argument exclusively to H .

Scheduled Games. The central idea is to replace the query-dependent milestones of the classical proof by a fixed schedule chosen independently of the adversary's execution. We partition the second phase into short blocks and associate each intermediate key k_i with a predetermined activation time. Before this time, the corresponding oracle value is replaced by an independent value; afterwards, it agrees with the value needed to reveal the next puzzle. This yields a sequence of scheduled games interpolating between the real execution and one in which the final challenge remains hidden until the adversary terminates. Crucially, adjacent games differ at only one oracle point over a fixed prefix, allowing their distinguishing gap to be controlled by the adversary's query mass at that point.

Concretely, let $L = T_1 - 1$ and pad the quantum adversary's second phase to a deterministic depth $D = o(T_0 T_1)$. At negligible statistical cost, we condition the L intermediate keys to be pairwise distinct and different from the two challenge strings. We then sample each $\mathbf{ct}_{i+1}$ uniformly and define the corresponding target value $u_i = \mathbf{ct}_{i+1} \oplus Z_{i+1}$, so that setting $H(k_i) = u_i$ recovers the honest masking distribution. Finally, we partition the padded computation into deterministic blocks using the cuts

$$r_i = \left\lceil \frac{i(D+1)}{L} \right\rceil, \quad h = \max_i (r_i - r_{i-1}) \leq \left\lceil \frac{D+1}{L} \right\rceil.$$

We now define a sequence of games $G^0, \dots, G^L$. In game G^j , each target with index greater than j is active from the start, while each other target u_i becomes active after round r_i . Fresh-point adaptive reprogramming shows that the first game G^0 is negligibly close to the real plain TLP security game. In the final game G^L , the last target depends on the challenge and stays inactive until the adversary gives its output. Therefore, the quantum adversary's success probability in G^L is exactly $1/2$.

Aggregate and Localize the Query Mass. We next convert the distinguishing gap across the scheduled games into query mass on the intermediate keys. Since adjacent games differ only at one point k_j over a fixed prefix, a one-point QROM switching bound relates each hybrid gap to the adversary's cumulative query mass on k_j . Summing these gaps and applying Cauchy-Schwarz once yields a lower bound on the aggregate mass across all links, without incurring a separate square-root loss at each hybrid step. We then show that this mass cannot be concentrated entirely before the corresponding scheduled block: because k_j remains hidden

up to that point, its earlier query mass is negligible. Hence a non-negligible overall advantage implies noticeable query mass on some k_j within its short block $(r_{j-1}, r_j]$.

Formally, the games G^{j-1} and G^j differ only at k_j through cut r_j . Applying the time-dependent one-point switch bound to each adjacent pair, and then a single Cauchy-Schwarz inequality across all j , relates the overall distinguishing gap to the aggregate query mass on the intermediate keys. Thus, any non-negligible advantage implies significant aggregate mass, without incurring a separate square-root loss at each link.

It remains to localize this mass to the short blocks $(r_{j-1}, r_j]$. Recall that, in the collision-free game, after $\mathcal{A}_1$ outputs x_0, x_1 , the ordered chain keys are sampled uniformly without replacement from $0, 1^\lambda \setminus \{x_0, x_1\}$. Fix x_0, x_1 , all chain keys other than k_j , and all challenger randomness sampled independently of k_j . Conditioned on these values, k_j is uniform over

$$R_j = \{0, 1\}^\lambda \setminus (\{x_0, x_1\} \cup \{k_i : i \neq j\}), \quad |R_j| \geq N'.$$

For $j > 1$, the adversary's view through round r_{j-1} is independent of the choice of k_j within R_j . Indeed, $Z_j = \text{PGen}(\text{st}, k_j)$ appears only through the target $u_{j-1} = \text{ct}_j \oplus Z_j$: the exception at k_{j-1} remains inactive through round r_{j-1} , while that at k_j remains inactive through round r_j . Hence, for the input-mass function μ of any query slot in this prefix,

$$\mathbb{E}_{k_j \leftarrow R_j}[\mu(k_j)] = \frac{1}{|R_j|} \sum_{x \in R_j} \mu(x) \leq \frac{1}{|R_j|} \leq \frac{1}{N'}.$$

Since there are at most Q such query slots, the total query mass accumulated on k_j before its designated block satisfies $E_j \leq Q/N'$. For $j = 1$, the prefix is empty.

Reusable Challenge Embedding. We now turn the localized query mass into an attack on the reusable security of the underlying symmetric TLP. The reduction embeds a single reusable-security challenge at a randomly selected link J and simulates the scheduled game only up to the beginning of its corresponding short block $(r_{J-1}, r_J]$. It then runs the adversary through this block and measures the input register of a randomly selected oracle query. The preceding localization argument ensures that noticeable query mass on the embedded key translates into a noticeable probability of extracting it, and hence of identifying the hidden challenge bit. Crucially, the reduction executes only one short block after receiving its challenge, so its second-phase depth is $o(T_0)$, as required to contradict symmetric reusable security of the underlying symmetric TLP with setup.

Concretely, the reduction chooses a uniformly random block J and simulates G^J through round r_{J-1} . At this point, it submits two fresh candidate keys to the reusable-security challenger and embeds the resulting challenge puzzle at position J . It then continues the simulation only for the block $(r_{J-1}, r_J]$. Before entering the block, the reduction selects a uniformly random padded query slot and measures its query-input register immediately before the corresponding oracle

call. If the measurement returns the embedded candidate, the reduction recovers the challenge bit; the probability of instead measuring the unused candidate is negligible. Importantly, activating the scheduled oracle value at the block boundary is part of the exact definition of G^J , rather than an additional oracle-reprogramming step. The QROM switching bounds are therefore needed only for the real-to- G^0 transition and for comparisons between adjacent scheduled games.

It remains to verify that the resulting reduction satisfies the required complexity bounds. Its second phase has depth at most $h + O(1)$, where

$$\frac{h}{T_0} \leq 2 \frac{D+1}{T_0 T_1} + \frac{1}{T_0} = o(1).$$

Thus, the reduction has second-phase depth $o(T_0)$ and loses only polynomial factors in T_1 , the query width, and the query count. Finally, the compiler oracle H can be simulated exactly, for at most Q logical queries, using a secret $2Q$ -wise independent base function together with a reversible table storing the scheduled exceptions; the underlying oracles $\mathbf{H}_u$ are simply forwarded unchanged. This yields a polynomial-size reduction and completes the proof of QROM security of our compiler for every polynomially bounded T_1 , including the balanced choice $T_0 = T_1 = \sqrt{T}$.

A.2 The Quantum Random Oracle Model

In this section, we fix the QROM model and the query-mass tools needed for the analysis.

Definition 9 (QROM with Parallel Queries). *Let $H: \{0,1\}^\lambda \rightarrow \{0,1\}^m$ be a uniformly random function (for the compiler, m denotes the length of the canonical puzzle encoding). In the quantum random oracle model, every party may apply the unitary*

$$O_H: |x\rangle|y\rangle \mapsto |x\rangle|y \oplus H(x)\rangle,$$

on registers of its choice. In the parallel-query QROM [18], an algorithm proceeds in query rounds; in one round it applies $O_H^{\otimes w}$ to w query/response register pairs, so that the w queries of a round may not depend on each other's answers.

Throughout this appendix, H denotes the compiler oracle. We write $\mathbf{H}_u = (H_a)_{a \in [t]}$ for a possibly empty tuple of mutually independent random oracles used by the underlying TLP. The oracle H is independent of every oracle in $\mathbf{H}_u$.

Definition 10 (Parallel Quantum Depth). *A quantum oracle adversary is a family of quantum circuits over a fixed universal gate set together with parallel oracle gates $O_H^{\otimes w}$. Its computation is divided into rounds: each round comprises at most one layer of (up to w -parallel) oracle gates together with a local circuit of at most $p_0(\lambda)$ sequential gate-layers, for the fixed polynomial p_0 of the depth convention of [Section 3.1](#); a round may also be purely local. Its depth d is the number of rounds; its size is the total number of gates. We write $q \leq d$ for the number of query rounds and $Q \leq qw$ for the total number of oracle queries. Polynomial-size adversaries have $w, Q \leq \text{poly}(\lambda)$.*

When several oracle gates are present, one query layer may contain parallel calls to any of them. All such calls count toward circuit size and depth. In the compiler analysis below, Q , w , and query mass count only calls to the compiler oracle H unless stated otherwise.

Note that non-oracle computation is charged as well — a depth- d adversary has total gate-depth at most $d \cdot p_0(\lambda)$ — which is essential, since solving the underlying puzzles is plain computation. The per-round polynomial allowance is the parallel-time convention fixed in [Section 3.1](#): it makes the honest solver (T rounds of $\text{poly}(\lambda)$ work) and reduction algorithms (which simulate oracles at $\text{poly}(\lambda)$ cost per round) commensurable, and it is used consistently in the assumptions ([Definition 13](#)) and the theorems below, so that it strengthens hypotheses and weakens conclusions in the same measure.

Efficient Oracle Simulation. A reduction cannot store an exponentially large random-function table. For at most Q calls to the compiler oracle H , it samples a secret $2Q$ -wise independent function. It evaluates this function reversibly, as in Zhandry’s finite-independence simulation [\[36\]](#). The output probabilities of a Q -query algorithm have oracle degree at most $2Q$. Thus, the substitution is exact. The simulator adds a polynomial-size classical table of time-dependent exceptions. See [Lemma 10](#) and [Appendix A.6](#) for the formal statement, reversible implementation, and depth analysis.

Query Mass. The following deterministic quantities replace the query times in the classical ROM analysis of [Theorem 1](#).

Definition 11 (Query Mass). *Fix a game Γ (an interaction of a quantum oracle adversary with a challenger). Throughout, the mass is measured on query registers for the compiler oracle H . Number the query rounds globally across both phases of the adversary: rounds $1, \dots, q_1$ belong to $\mathcal{A}_1$ and rounds $(q_1 + 1), \dots, (q_1 + q_2)$ to $\mathcal{A}_2$; we set $q = (q_1 + q_2)$ and let $Q \leq qw$ count the queries of both phases. Let $\mathbf{rand}$ denote all classical randomness of the challenger and the adversary. For a point $x = x(\mathbf{rand})$ (possibly a function of $\mathbf{rand}$, such as a key k_i) and a query round s , let*

$$\mu_s^\Gamma(x) = \mathbb{E}_{\mathbf{rand}} \left[\|\Pi_{x(\mathbf{rand})} \psi_s(\mathbf{rand})\|^2 \right]$$

denote the squared amplitude of the round- s query registers on $|x(\mathbf{rand})\rangle$ (summed over the $\leq w$ parallel registers), averaged over $\mathbf{rand}$. The cumulative query mass of x up to round r is

$$W^\Gamma(x; r) = \sum_{s \leq r} \mu_s^\Gamma(x), \quad W^\Gamma(x) = W^\Gamma(x; q).$$

For an interval I of rounds we write $W^\Gamma(x; I) = \sum_{s \in I} \mu_s^\Gamma(x)$. The schedule in [Appendix A.4](#) uses full circuit depth. Thus, the numbering includes each local-only depth round. We assign zero mass to its empty oracle lanes.

For fixed Γ , x , and r , the quantity $W^\Gamma(x; r)$ is a deterministic real number obtained by averaging over the randomness of the game and the adversary. This allows us to reason about query mass without measuring the adversary's queries. In the proof below, we use these quantities with a fixed, adversary-independent schedule of depth cuts: negligible mass before the designated block captures that a hidden key has not been meaningfully queried, while noticeable mass within a short block can be converted, via random-query measurement, into an extraction probability. Since the intermediate keys $k_1, \dots, k_{T_1-1}$ are sampled only after $\mathcal{A}_1$ halts, their phase-1 query mass is negligible and is absorbed into the error terms below.

The One-Way-to-Hiding Lemma. We import the one-way-to-hiding lemma in the following form; it is the special case of the semi-classical O2H theorem of Ambainis, Hamburg and Unruh for reprogramming at a single point, and in the adaptive variant of Grilo et al.

Lemma 4 (O2H, One Point [34,4,24]). *Let Γ, Γ' be two games that are identical except that a value handed to the adversary as x^* 's oracle output is decoupled: in Γ the challenger uses $H(x^*)$, in Γ' an independent uniform string (equivalently, H is reprogrammed at the single point x^* , possibly at an intermediate round r_0 , before which the challenger has not used $H(x^*)$). Then, for every event E defined on the adversary's output,*

$$|\Pr_\Gamma[E] - \Pr_{\Gamma'}[E]| \leq 2\sqrt{(Q+1) \cdot W^{\Gamma'}(x^*)},$$

and the same bound holds with the total variation distance of the adversary's output distributions on the left. For two fixed time-dependent oracles, use the direct telescoping form in Lemma 7. The two oracles can differ at one point in a specified set of slots. A scheduled activation inside one fixed auxiliary game is not an indistinguishability step. Treat one round of w parallel queries as w sequential slots with no intermediate computation. The value Q counts all lanes.

Two further elementary lemmas replace the classical “ k_i is a uniform string independent of the view” step and the classical detection of queries.

Lemma 5 (Mass on a Hidden Key). *Let Γ be a game in which a key $x^* \leftarrow \{0,1\}^\lambda$ is sampled uniformly and the challenger's entire message transcript is independent of x^* ; the only x^* -dependence of the game is that oracle queries at x^* are answered by the (possibly challenger-correlated) value $H(x^*)$. Then*

$$W^\Gamma(x^*) \leq \frac{Q}{2^\lambda} + \frac{5Q^2\sqrt{w}}{2^{\lambda/2}} =: \beta,$$

where W is the averaged mass of Definition 11 (the expectation over x^* is part of W). The same bound holds, up to a factor 2, when x^* is uniform on a subset $S \subseteq \{0,1\}^\lambda$ with $|S| \geq 2^\lambda - T_1$ — the case arising after conditioning on key-distinctness.

Proof. Let Γ_0 be the modified game in which queries at x^* are answered by an independent random function value (i.e. the point x^* carries no planted correlation). In Γ_0 the entire execution is independent of x^* , so for each round the expected mass on x^* , averaged over x^* , is at most $w \cdot 2^{-\lambda}$, whence $W^{\Gamma_0}(x^*) \leq Q 2^{-\lambda}$. For each fixed value of the randomness, the executions of Γ and Γ_0 differ only in the oracle value at the single point x^* ; by the BBBV hybrid argument, the Euclidean distance of the joint states after round s is at most $2 \sum_{s' \leq s} \sqrt{w \mu_{s'}^{\Gamma_0}(x^*)} \leq 2 \sqrt{w Q W^{\Gamma_0}(x^*)} =: \eta$ (the $\sqrt{w}$ accounts for the w parallel registers of a round), so each round's mass changes by at most $2\eta\sqrt{\mu_s} + \eta^2$, giving, after summing over the $\leq Q$ rounds, applying Cauchy-Schwarz to $\sum_s \sqrt{\mu_s}$, taking the expectation over x^* pointwise (the hybrid bound is expressed entirely in terms of Γ_0 -masses, whose average is known), and using Jensen,

$$W^\Gamma(x^*) \leq \frac{Q}{2^\lambda} + 4Q \sqrt{\frac{w Q \cdot Q}{2^\lambda}} + 4w Q^3 2^{-\lambda} \leq \frac{Q}{2^\lambda} + \frac{5Q^2 \sqrt{w}}{2^{\lambda/2}}.$$

(The exponent $\lambda/2$ is far from tight but suffices: the right-hand side is negligible for $Q, w = \text{poly}(\lambda)$.)

Lemma 6 (Extraction by a Random Query Measurement). *Let Γ be a game, x^* a point, and I an interval of query rounds containing at most Q_I queries in total. Consider the experiment that runs Γ , picks one of the Q_I query registers of the interval I uniformly at random, and measures it in the computational basis just before the corresponding oracle call. The probability that the outcome equals x^* is at least $W^\Gamma(x^*; I)/Q_I$, and the states of all rounds before the measured one are undisturbed.*

Proof. Averaging: the probability equals $\frac{1}{Q_I} \sum \langle \Pi_{x^*} \rangle$ over the interval's query registers, which is $W^\Gamma(x^*; I)/Q_I$; the execution up to the measured query is not touched.

Both [Lemma 4](#) and [Lemma 6](#) are applied below with the averaged masses of [Definition 11](#), at points that are functions of the challenger's randomness. [Lemma 6](#) holds verbatim in this form by linearity of expectation. [Lemma 4](#) holds with W replaced by its average by Jensen's inequality, $\mathbb{E}[\sqrt{W}] \leq \sqrt{\mathbb{E}[W]}$, which moves the error term in the favorable direction; we have checked that every application in this section carries the square root on the bounding side, so this substitution is sound throughout.

A.3 Post-Quantum Security Definitions for TLP Variants

We next formulate the post-quantum security notions for different TLP variants that are needed for analyzing the compiler in the QROM. Since the TLP schemes themselves remain classical, their syntax, correctness, and efficiency requirements are unchanged; only the adversarial model must be strengthened. In particular, we allow polynomial-size quantum adversaries with the appropriate depth restriction

and, whenever a scheme uses a random oracle, grant the adversary quantum superposition access to it. We formalize this first for the security of the compiled plain TLP, and then for the reusable security of the underlying symmetric TLP with setup. These are the two notions connected by the QROM compiler theorem proved in the following subsections.

Post-Quantum Plain TLP. For plain TLP without setup/preprocessing, the syntactic (Definition 1), correctness (Definition 2) and efficiency definitions remain unchanged. We present the post-quantum version of its security definition below.

Definition 12 (Post-Quantum Plain TLP Security in the QROM). *Let $\mathbf{H}$ be a possibly empty finite tuple of mutually independent random oracles, with domains and ranges fixed by the scheme. A plain-model TLP $(\text{PGen}^*, \text{PSolve}^*)$ whose algorithms may access $\mathbf{H}$ is post-quantum secure in the QROM if there exists a polynomial $\bar{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \bar{T}(\cdot)$ and every polynomial-size quantum adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in which the depth (Definition 10) of $\mathcal{A}_2$ is $o(T)$, there is a negligible $\mu(\cdot)$ with, for all λ ,*

$$\Pr \left[\begin{array}{l} \mathbf{H} \leftarrow \text{RO}, \\ b \leftarrow \mathcal{A}_2^{|\mathbf{H}|}(Z^*, \mathbf{z}) : (x_0, x_1, \mathbf{z}) \leftarrow \mathcal{A}_1^{|\mathbf{H}|}(1^\lambda, T(\lambda)) \\ b \leftarrow \{0, 1\}, \quad Z^* \leftarrow (\text{PGen}^*)^{\mathbf{H}}(\lambda, T(\lambda), x_b) \end{array} \right] \leq \frac{1}{2} + \mu(\lambda),$$

Here $x_0, x_1 \in \{0, 1\}^\lambda$, $\mathbf{z}$ is a quantum state, and $|\mathbf{H}|$ denotes quantum access to each oracle in the tuple.

Post-Quantum Symmetric TLP with Setup. Again, for symmetric TLP with setup, the syntactic (Definition 5), correctness (Definition 6) and efficiency (Definition 8) definitions remain unchanged. We present the post-quantum version of its symmetric reusable security below.

Definition 13 (Post-Quantum Symmetric Reusable Security). *Let $\mathbf{H}_u$ be a possibly empty finite tuple of mutually independent random oracles, with domains and ranges fixed by the scheme. A symmetric TLP has setup algorithm PSetup , generation algorithm PGen , and solver PSolve . Its solutions have length λ . Its three algorithms may make classical queries to the same tuple $\mathbf{H}_u$. It is post-quantum reusable secure if the following condition holds. There is a polynomial $\bar{T}$. For each polynomial $T_0 \geq \bar{T}$ and each polynomial-size quantum adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$, there is a negligible function μ . The adversary can be non-uniform, and its second phase has depth $o(T_0)$. The function μ satisfies*

$$\Pr \left[\begin{array}{l} \mathbf{H}_u \leftarrow \text{RO}, \quad \text{st} \leftarrow \text{PSetup}^{\mathbf{H}_u}(1^\lambda, T_0), \\ \beta' = \beta : (\kappa_0, \kappa_1, \zeta) \leftarrow \mathcal{B}_1^{|\mathbf{H}_u|, \mathcal{O}}(1^\lambda, T_0), \\ \kappa_0, \kappa_1 \in \{0, 1\}^\lambda, \quad \beta \leftarrow \{0, 1\}, \quad Z^* \leftarrow \text{PGen}^{\mathbf{H}_u}(\text{st}, \kappa_\beta; r^*), \\ \beta' \leftarrow \mathcal{B}_2^{|\mathbf{H}_u|}(Z^*, \zeta) \end{array} \right] \leq \frac{1}{2} + \mu(\lambda).$$

Here ζ can be any quantum state. The challenger transfers this state between the two phases and does not copy it. Both phases have quantum access to the same sampled tuple $\mathbf{H}_u$. The oracle $\mathcal{O}$ accepts only classical messages and returns $\mathcal{O}(s) = \text{PGen}^{\mathbf{H}_u}(\text{st}, s; r)$ with fresh independent coins r on each call. The challenge uses fresh independent coins r^* . Puzzle generation does not modify st . The algorithm $\mathcal{B}_2$ cannot access $\mathcal{O}$, but it retains access to $\mathbf{H}_u$. All of its oracle calls count toward its size and depth. Only the depth of $\mathcal{B}_2$ is restricted.

The tuple $\mathbf{H}_u$ is empty for a random-oracle-free underlying TLP. In the isogeny instantiation, it contains $\mathbf{H}_{\text{walk}}$ and $\mathbf{H}_{\text{pf}}$. The compiler uses an additional independent oracle H . The reduction forwards every query to $\mathbf{H}_u$ and internally simulates only H .

A.4 The QROM Analogue of Theorem 1

We now formally prove the QROM analogue of Theorem 1. The correctness and efficiency analyses remain unchanged from Section 4.2, so the main task is to establish post-quantum security. Concretely, we show that any polynomial-size quantum adversary breaking the compiled TLP with second-phase depth $o(T_0 T_1)$ can be transformed into an adversary that violates the post-quantum reusable security of the underlying symmetric TLP with setup, while using only $o(T_0)$ depth after receiving its challenge. The proof implements the delayed-activation strategy outlined in Appendix A.1: we pass through a sequence of scheduled QROM games, convert their distinguishing gaps into localized query mass, and finally use that mass to embed and recover a reusable-security challenge.

Theorem 6 (Our Compiler in the QROM). *[Formal Version of Theorem 2] Let T_0 and T_1 be polynomially bounded, and let $T = T_0 T_1$. Suppose that $T_0 \rightarrow \infty$ and eventually enters the security range of the underlying scheme. Assume that $(\text{PSetup}, \text{PGen}, \text{PSolve})$ is a symmetric TLP with setup satisfying Definitions 6, 8 and 13, relative to a possibly empty tuple $\mathbf{H}_u$ of independent quantum random oracles. Model the compiler oracle H as an additional quantum random oracle independent of $\mathbf{H}_u$. Then the compiler of Section 4.2 is correct and post-quantum secure in the QROM relative to the joint oracle family $(\mathbf{H}_u, H)$ for every polynomially bounded T_1 . Its generation time is $(T_0 + T_1) \text{poly}(\log T, \lambda)$, while its solving time is $T_0 T_1 \text{poly}(\lambda)$. In particular, $T_0 = T_1 = \sqrt{T}$ gives generation time $\sqrt{T} \text{poly}(\log T, \lambda)$ and solving time $T \text{poly}(\lambda)$.*

Proof of QROM Security. Fix a polynomial-size quantum adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Suppose its second phase has depth $D = o(T_0 T_1)$. Pad its circuit to deterministic depth D . The padding includes local-only rounds. The depth D counts all local computation and all oracle layers. Let Q_1 be the number of calls to the compiler oracle H from $\mathcal{A}_1$. Let Q bound the calls to H from both phases, and let $w \geq 1$ be the maximum number of parallel H -query lanes in one second-phase round. Queries to $\mathbf{H}_u$ count toward circuit size and depth but not toward these three quantities or the query masses below. Let ε be the advantage in the compiled plain TLP security game.

First, suppose that $T_1 \geq 2$. Set $L = T_1 - 1$ and $N = 2^\lambda$. Require the L chain keys to be pairwise distinct. Also require them to differ from the two challenge strings that $\mathcal{A}_1$ gives. We can sample these keys without replacement from the remaining domain. These requirements change the real game by at most

$$\Delta_{\text{coll}} \leq \frac{(L+2)^2}{2N}, \quad N' = N - (L+4).$$

For polynomial L , we eventually have $N' > 0$, and the loss is negligible. We do not need a condition on collisions between outputs of PGen .

For $i \in [L]$, let $Z_i \leftarrow \text{PGen}(\text{st}, k_i)$ and $Z_{L+1} \leftarrow \text{PGen}(\text{st}, x_b)$, where (x_0, x_1) are the challenge strings from $\mathcal{A}_1$. Sample each ct_{i+1} uniformly. Define the target

$$u_i = \text{ct}_{i+1} \oplus Z_{i+1}.$$

The assignment $H(k_i) = u_i$ gives ct_{i+1} the honest mask distribution.

The Schedule and the Games. Define deterministic cuts

$$r_0 = 0, \quad r_i = \left\lceil \frac{i(D+1)}{L} \right\rceil \quad (i \in [L]), \quad h = \max_{i \in [L]} (r_i - r_{i-1}) \leq \left\lceil \frac{D+1}{L} \right\rceil.$$

For $j \in \{0, \dots, L\}$, game $\mathbf{G}^j$ lets $\mathcal{A}_1$ query a base random function F . After $\mathcal{A}_1$ finishes, the game runs $\mathcal{A}_2$ with the time-dependent oracle

$$H_t^{(j)}(x) = \begin{cases} u_i, & x = k_i \text{ and } (i > j \text{ or } t > r_i), \\ F(x), & \text{otherwise,} \end{cases} \quad t \in [D].$$

All games use the same sampled tuple $\mathbf{H}_u$ and answer queries to it identically. Only the compiler oracle H is replaced by the scheduled oracle $H_t^{(j)}$. The distinctness condition makes the exception table unambiguous. In $\mathbf{G}^0$, every chain target is active when the second phase starts. In $\mathbf{G}^L$, every target follows the schedule. Games $\mathbf{G}^{j-1}$ and $\mathbf{G}^j$ differ only at k_j through round r_j . Empty query lanes and the formal round $D+1$ have mass zero.

For a second-phase depth round t and lane $a \in [w]$, let $\mu_{t,a}^{\mathbf{G}}(x)$ be the expected squared amplitude on input x immediately before that oracle call. Set this value to zero when the lane is empty. For a set I of depth rounds, write

$$W^{\mathbf{G}}(x; I) = \sum_{t \in I} \sum_{a=1}^w \mu_{t,a}^{\mathbf{G}}(x).$$

This is the query mass from [Definition 11](#). Here, all depth rounds are indexed, not only rounds with an oracle gate.

Lemma 7 (Time-Dependent One-Point Switch). *Suppose that two quantum-oracle executions differ only in the answer at one point x . This difference occurs in a specified set S of query slots. Let M be the total query mass on x in these slots for one fixed reference execution. Then the statistical distance between the output*

distributions is at most $2\sqrt{(Q+1)M}$. The result also holds when a deterministic depth schedule selects S . The two oracles must have the same classical updates outside S . Identical auxiliary quantum random oracles may be interleaved with these calls.

Proof. Replace the different oracle calls one slot at a time. Suppose that the input mass on x is μ in one slot. A change to the answer changes the Euclidean norm of the joint state by at most $2\sqrt{\mu}$. Apply the standard BBBV telescoping hybrid. Then apply Cauchy–Schwarz to at most Q calls and use data processing for the final measurement. This gives the bound. Deterministic changes to the common circuit between calls do not affect the argument. The term $+1$ also covers a terminal comparison. If the point or either execution depends on classical randomness, apply the argument after conditioning on that randomness. Then average and use Jensen’s inequality. Thus, M may be the averaged query mass of [Definition 11](#). The same conditioning argument covers identical auxiliary oracles.

Couple the real game and G^0 so that the second phase sees the same final function and masks. The games differ in one way. Before the second phase of G^0 , resample the function at the L fresh keys selected after $\mathcal{A}_1$. For all sufficiently large λ , $Q_1/N' < 1$. Condition on $\mathbf{H}_u$ and apply tight adaptive reprogramming only to H [\[24\]](#) at these conditioned-uniform points. This gives

$$|\Pr[\text{Real wins}] - \Pr[G^0 \text{ wins}]| \leq \eta_{\text{AR}}, \quad \eta_{\text{AR}} = \frac{3}{2}L\sqrt{\frac{Q_1}{N'}} + \Delta_{\text{coll}}.$$

The bound is negligible for polynomial L and Q_1 . At the other endpoint, only the target $u_L = \text{ct}_{L+1} \oplus Z_{L+1}$ depends on b . In G^L , this target stays inactive until $\mathcal{A}_2$ gives its output. The value ct_{L+1} is uniform. Therefore,

$$\Pr[G^L \text{ wins}] = \frac{1}{2}.$$

Let $p_j = \Pr[G^j \text{ wins}]$, $M_j = W^{G^j}(k_j; [1, r_j])$ and $\alpha_j = |p_{j-1} - p_j|$. By [Lemma 7](#), $\alpha_j \leq 2\sqrt{(Q+1)M_j}$. For $\varepsilon > \eta_{\text{AR}}$, use the two endpoints and apply Cauchy–Schwarz once. This is the only case that matters when ε is non-negligible. We obtain

$$\sum_{j=1}^L M_j \geq \frac{(\varepsilon - \eta_{\text{AR}})^2}{4(Q+1)L}.$$

This step prevents a new square-root loss at each chain link.

Lemma 8 (Localization to Short Blocks). *Assume $\varepsilon > \eta_{\text{AR}}$. Define*

$$E_j = W^{G^j}(k_j; [1, r_{j-1}]), \quad V_j = W^{G^j}(k_j; (r_{j-1}, r_j]).$$

Then $E_1 = 0$, $E_j \leq Q/N'$ for $j > 1$, and

$$\sum_{j=1}^L V_j \geq \frac{(\varepsilon - \eta_{\text{AR}})^2}{4(Q+1)L} - \frac{LQ}{N'}.$$

Proof. The first prefix is empty. For $j > 1$, condition on (x_0, x_1) , all chain keys other than k_j , and all classical randomness sampled independently of k_j . Conditional on these data, k_j is uniform over a residual set R_j with $|R_j| \geq N'$. Moreover, the public input and the execution of G^j through round r_{j-1} do not depend on which point of R_j is designated as k_j . Indeed, ct_j is uniform, Z_j occurs only in $u_{j-1} = \text{ct}_j \oplus Z_j$, and the exception at k_{j-1} remains inactive through round r_{j-1} . The exception at k_j remains inactive through round r_j . Thus, for each prefix query slot with input-mass function μ ,

$$\mathbb{E}_{k_j \leftarrow R_j}[\mu(k_j)] = \frac{1}{|R_j|} \sum_{x \in R_j} \mu(x) \leq \frac{1}{|R_j|} \leq \frac{1}{N'}.$$

There are at most Q prefix query slots, so $E_j \leq Q/N'$. Averaging over the conditioning gives the same bound. Now sum $M_j = E_j + V_j$ and use (A.4). This proves the claim.

Lemma 9 (Reduction to Single Reusable Challenge). *Assume $\varepsilon > \eta_{\text{AR}}$. There is a polynomial-size quantum adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against Definition 13. Its advantage satisfies*

$$\text{Adv}(\mathcal{B}) \geq \frac{(\varepsilon - \eta_{\text{AR}})^2}{8(Q+1)L^2wh} - \frac{Q}{2whN'} - \frac{1}{2N'}.$$

Under the round convention of Definition 10, its second phase, i.e., $\mathcal{B}_2$ has depth at most $(h + O(1))$.

Proof. Throughout both phases, $\mathcal{B}$ forwards every query to $\mathbf{H}_u$ to its own underlying-oracle gates and internally simulates only the compiler oracle H . The underlying challenger runs PSetup before $\mathcal{B}_1$, whereas the compiled generator runs it after $\mathcal{A}_1$. Moving this hidden setup earlier preserves the joint distribution: $\mathcal{A}_1$ receives no setup-dependent data, the setup coins are independent, and read-only oracle queries do not alter $\mathbf{H}_u$. $\mathcal{B}_1$ proceeds via the following steps.

1. It samples $J \leftarrow [L]$ and simulates $\mathcal{A}_1$ with the base compiler quantum random oracle.
2. After $\mathcal{A}_1$ gives $(x_0, x_1, \mathbf{z})$, it selects an independent compiled-game bit b' .
3. It samples all chain keys except k_J . It also samples the masks $\text{ct}_2, \dots, \text{ct}_{L+1}$ uniformly.
4. It uses its classical $\text{PGen}(\text{st}, \cdot)$ oracle to obtain every required puzzle except Z_J . This includes $Z_{L+1} \leftarrow \text{PGen}(\text{st}, x_{b'})$.
5. It starts $\mathcal{A}_2$ and simulates G^J through round r_{J-1} . If $J = 1$, it does not start $\mathcal{A}_2$ because the public first puzzle is not available.
6. After the prefix, it samples ordered distinct strings κ_0, κ_1 uniformly without replacement from $\{0, 1\}^\lambda \setminus (\{x_0, x_1\} \cup \{k_i : i \neq J\})$. It submits them as its reusable-security challenge messages.
7. It transfers its complete state to $\mathcal{B}_2$. The state contains the unmeasured $\mathcal{A}$ state, the secret base-function seed, the active exception table, J , the masks, and all classical randomness. It does not copy a quantum state.

The prefix needs neither k_J nor Z_J . The point k_J is inactive. Also, the only target that contains Z_J is u_{J-1} , which stays inactive through the cut.

The underlying challenger selects β and sends $Z^* \leftarrow \text{PGen}(\text{st}, \kappa_\beta)$ to $\mathcal{B}_2$. Then $\mathcal{B}_2$ proceeds via the following steps.

1. If $J > 1$, it activates the known predecessor point k_{J-1} with target $\text{ct}_J \oplus Z^*$. If $J = 1$, it installs Z^* as the public first puzzle and initializes $\mathcal{A}_2$.
2. It selects one of the wh padded oracle slots in block $(r_{J-1}, r_J]$ uniformly.
3. It executes the block up to that slot. Immediately before the corresponding oracle call, it measures the query-input register. An empty slot gives no outcome.
4. It outputs 0 on outcome κ_0 and 1 on outcome κ_1 . For any other outcome, it outputs a uniform bit.

The first step is the specified boundary transition in $\mathbf{G}^J$. It changes only future oracle gates and leaves the current joint state unchanged. Thus, it is an exact game simulation, not an adaptive reprogramming comparison. The embedded key is $k_J = \kappa_\beta$. This point stays inactive throughout the block. Therefore, the simulator does not need to know β or program an unknown point.

Conditioned on $J = j$, the probability of measuring κ_β is $V_j/(wh)$. Let $U = \kappa_{1-\beta}$. Condition on $J = j$, β , $k_j = \kappa_\beta$, all other chain keys, the selected padded slot, and all classical randomness sampled independently of U . Then U is uniform over a residual set R with $|R| \geq N'$, and the simulated query state before the selected measurement does not otherwise use U . If $p(x)$ is the input distribution in that slot, with $p \equiv 0$ for an empty slot, then

$$\Pr[\text{outcome} = U \mid \text{conditioning}] = \frac{1}{|R|} \sum_{x \in R} p(x) \leq \frac{1}{|R|} \leq \frac{1}{N'}.$$

Averaging over the conditioning gives the same bound. Therefore,

$$\text{Adv}(\mathcal{B}) \geq \frac{1}{2Lwh} \sum_{j=1}^L V_j - \frac{1}{2N'}.$$

Substitute (8) to prove (9). The second phase runs at most one block, a constant-depth setup, and one measurement.

Lemma 10 (Efficient Dynamic Quantum Random Oracle Simulation).

Suppose a computation makes at most Q logical calls to the compiler quantum random oracle H . The base function and each scheduled oracle in (A.4) have an exact polynomial-size reversible implementation. One logical oracle call has depth $\text{polylog}(\lambda, Q, L, m)$, where m is the oracle output length. The simulator does not reveal its private seed to the simulated adversary. The simulator forwards $\mathbf{H}_u$ and does not include calls to those oracles in Q .

Proof. Use a secret $2Q$ -wise independent base function. Add the active exception table to this function. The polynomial method makes the substitution exact for

Q logical queries [36]. Reversible equality tests and a balanced table multiplexer implement each exception. The implementation computes and un-computes its temporary values cleanly. See [Appendix A.6](#) for the full construction and degree argument. For fixed $\mathbf{H}_u$, its gates are ordinary inter-query operations in this argument. Averaging over $\mathbf{H}_u$ preserves exactness.

Proof of Theorem 6. We now finish the proof of [Theorem 6](#). First, suppose that $T_1 \geq 2$. The quantities Q, L, w , and h are polynomially bounded. Since L is polynomially bounded, $N' = 2^\lambda - (L + 4) \geq 2^{\lambda-1}$ for all sufficiently large λ . Hence η_{AR} and both terms in (9) that contain $1/N'$ are negligible. The denominator $8(Q + 1)L^2wh$ in the first term is polynomially bounded. Thus, if ε is non-negligible, then $\varepsilon - \eta_{\text{AR}}$ and the advantage of $\mathcal{B}$ are non-negligible. Also,

$$\frac{h}{T_0} \leq \frac{D + 1}{T_0(T_1 - 1)} + \frac{1}{T_0} \leq 2\frac{D + 1}{T_0 T_1} + \frac{1}{T_0} = o(1).$$

Under the round convention of [Definition 10](#), the fixed local-work allowance includes the polylogarithmic evaluator from [Lemma 10](#). Thus, $\mathcal{B}_2$ has depth $o(T_0)$. This contradicts [Definition 13](#).

If $T_1 = 1$, the compiled challenge is one underlying puzzle, and there is no hash-chain link. A direct reduction forwards $\mathbf{H}_u$ and simulates the independent compiler quantum random oracle for $\mathcal{A}$. It transfers the state once and embeds the underlying challenge. Its second-phase depth is $D = o(T_0)$. If $T_1(\lambda)$ alternates between 1 and larger values, the reduction selects the applicable case for each λ . This proves security in all cases.

A.5 Discussion

Scope. [Theorem 6](#) assumes post-quantum reusable security of the underlying TLP relative to any quantum random oracles used by that TLP; see [Definition 13](#). The compiler oracle H is independent of these underlying oracles. The reduction forwards the underlying oracles unchanged and simulates and schedules only H . The concrete isogeny TLP uses the independent oracles $\mathbf{H}_{\text{walk}}$ and $\mathbf{H}_{\text{pf}}$ and satisfies this hypothesis under [Assumption 1](#), by [Theorem 5](#).

Approximate Oracle Changes. The comparison of the real game with $\mathbf{G}^0$ uses fresh adaptive reprogramming. Each comparison $\mathbf{G}^{j-1} \leftrightarrow \mathbf{G}^j$ is an approximate one-point oracle switch. The bound in [Lemma 7](#) controls this hybrid. The activation of k_{J-1} at the scheduled boundary is different. Inside the fixed game $\mathbf{G}^J$, this activation is an exact simulation. The game uses a time-dependent oracle, and the change can be detectable.

Depth Units. Equation (A.4) counts all second-phase depth rounds. This count includes local-only rounds. For raw gate depth, the reduction needs the following condition instead of the round convention from [Definition 10](#): $h c_{\text{sim}}(\lambda) = o(T_0)$. Here, c_{sim} is the depth of one clean dynamic-oracle evaluation. The implementation in [Appendix A.6](#) has polylogarithmic depth. The fixed polynomial local-work allowance is large enough for this depth.

Collisions and Boundary Cases. Sampling the chain keys without replacement is a proof device. Its cost is $\Delta_{\text{coll}} = O((T_1 + 1)^2/2^\lambda)$. The reduction samples its two candidates so that they differ from the other keys and challenge strings. We do not need to condition on collisions between outputs of PGen. The scheduled proof covers $T_1 = 2$ directly. The direct reduction above covers $T_1 = 1$. Repeated cuts and zero-query blocks cause no problem when $L > D$ because empty slots have zero mass.

On Query Complexity. The compiler oracle H controls the order in which the underlying puzzles become visible. The delay at each link comes from Definition 13. An unbounded-depth algorithm can solve the chain directly. Thus, QROM query-mass bounds give the extraction step. Reusable security gives the time lower bound for each link.

A.6 Dynamic Finite-Family Simulation of the QROM

The preceding reduction was described using an ideal random function together with a small collection of scheduled exceptions. To complete the proof, we must verify that this oracle can be implemented by a polynomial-size quantum circuit without changing the adversary's view. We do so by replacing the ideal base function with a secret sufficiently independent function family, which is exact for the bounded number of quantum queries made by the adversary, and maintaining the scheduled exceptions in a reversible lookup table. This yields an efficient and exact implementation of the compiler oracle H required by the reduction; the independent underlying oracles $\mathbf{H}_u$ are simply forwarded unchanged. The corresponding implementation details for Lemma 10 are described below.

Let Q be the total number of logical calls to the compiler oracle H from the simulated adversary. This count includes both phases and each parallel H -query lane. The reduction forwards the underlying tuple $\mathbf{H}_u$ unchanged. The challenger samples the ct_i and the targets u_i directly. Thus, it does not make additional classical evaluations of the base function.

The Finite Base Family. Let $n \geq \max\{\lambda, m\}$, and identify n -bit strings with $\mathbb{F}_{2^n}$. Select independent uniform coefficients $a_0, \dots, a_{2Q-1} \in \mathbb{F}_{2^n}$. Define

$$P_\sigma(X) = \sum_{d=0}^{2Q-1} a_d X^d, \quad F_\sigma(x) = \text{trunc}_m(P_\sigma(\text{enc}(x))).$$

At any $2Q$ distinct points, a random degree- $(2Q - 1)$ polynomial has uniform and independent values. Truncation preserves this property. Thus, $F_\sigma : \{0, 1\}^\lambda \rightarrow \{0, 1\}^m$ is a $2Q$ -wise independent function.

Dynamic Exceptions. At each depth round, the simulator stores a classical list of distinct triples (k_i, u_i, a_i) . The bit a_i shows whether the exception is active. A clean reversible oracle call on (x, y) does the following steps.

1. Compute $F_\sigma(x)$ in an ancilla register.
2. Compute the equality bits $[x = k_i]$ in parallel. AND them with the public active bits a_i .
3. Use a balanced multiplexer. Select u_i if an active equality bit is one. Otherwise, select $F_\sigma(x)$.
4. XOR the selected value into y . Then reverse all temporary computations.

Because the keys are distinct, at most one exception applies. An activation at a scheduled boundary changes only the classical control bits for future oracle gates. It does not measure or modify the current state of the adversary.

Exactness. After at most Q quantum queries, each final amplitude has degree at most Q in the truth-table indicator variables. Each output probability has total degree at most $2Q$. Therefore, a $2Q$ -wise independent function preserves every output probability of a uniform random function exactly [36]. The equality tests and target multiplexer contain no truth-table variables. The same is true for the inter-query unitaries, activation schedule, and transfer of the simulator state. Thus, these operations do not increase the degree. The seed σ stays in the private workspace of the reduction. The reduction does not give, measure, or inspect this seed for the simulated adversary.

For each fixed choice of $\mathbf{H}_u$, calls to those oracles are inter-query operations that contain no truth-table variables of H . The argument above is therefore exact after conditioning on $\mathbf{H}_u$, and averaging over that tuple preserves exactness.

Suppose a different implementation makes R classical evaluations of F_σ outside the simulated adversary. In this case, use a $2(Q + R)$ -wise independent family. A $4Q$ -wise independent family is also a valid conservative choice. Clean computation and uncomputation of one logical oracle call do not count as two adversarial oracle queries.

Size and Depth. Use parallel powering to form all powers needed for $P_\sigma(x)$. Multiply them by the coefficients in parallel. Then use a balanced tree to add the results. Standard reversible binary-field operations have polynomial size and $\text{polylog}(\lambda, Q, L, m)$ depth. The same bounds apply to the balanced equality tests and the exception multiplexer. Computation, answer copying, and uncomputation change only constant factors. Thus, one logical QRO layer fits in the fixed polynomial local-work allowance of [Definition 10](#). In a raw gate-depth model, charge the factor in [Appendix A.5](#) explicitly.

A.7 Proof of [Theorem 5](#) (Post-Quantum Security in the QROM)

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a polynomial-size quantum adversary with advantage ϵ and second-phase depth $d = o(T)$. Let q_{PG} bound its puzzle-generation queries and q_H its $\mathbf{H}_{\text{pf}}$ queries across both phases, counting parallel query registers separately, and set $Q = q_H + q_{\text{PG}}$. We model the two hash functions $\mathbf{H}_{\text{walk}}$ and $\mathbf{H}_{\text{pf}}$ defined above as independent quantum random oracles. Both adversarial phases have quantum access to them.

We reuse the walk embedding and puzzle-generation simulation from the ROM proof of [Theorem 4](#). Namely, $\mathcal{D}_1$ uses $q = q_{\text{PG}}$ prechallenge isogenies, programs H_{walk} before the first adversarial query so that the public seed generates the supplied walk, and answers every classical puzzle-generation query by computing the corresponding pushforward. The distribution and first-phase running-time analysis are exactly as in that proof. The only changes are that the two random oracles are implemented coherently and the state $\mathbf{z}$ passed between the adversarial phases may be quantum.

As in the ROM proof, after receiving the fresh challenge isogeny ψ , $\mathcal{D}_2$ cannot compute its pushforward within depth $o(T)$. It instead samples independent $b \leftarrow \{0, 1\}$ and $u \leftarrow \{0, 1\}^\lambda$ and runs $\mathcal{A}_2(Z, \mathbf{z})$, where

$$Z = (\text{pp}, \psi, u \oplus s_b).$$

Call this game G_{sim} . Its independent mask hides b perfectly, so

$$\Pr_{G_{\text{sim}}} [b' = b] = \frac{1}{2}.$$

For the analysis only, sample ψ before the first phase and reveal it at the prescribed time. Let F'' be the codomain of $[\phi]_*\psi$, and define $\tilde{H}_{\text{pf}}$ by reprogramming $H_{\text{pf}}(F'')$ to u while leaving every other value unchanged. The game using $\tilde{H}_{\text{pf}}$ throughout both phases and all puzzle-generation evaluations is exactly the real security experiment: F'' is independent of H_{pf} and u , so $\tilde{H}_{\text{pf}}$ is a random oracle and $u = \tilde{H}_{\text{pf}}(F'')$. Let W be the total expected query mass on F'' among the Q counted calls to H_{pf} in G_{sim} , including the classical puzzle-generation evaluations. The one-point O2H lemma ([Lemma 4](#)) gives

$$\epsilon \leq 2\sqrt{(Q+1)W}.$$

Queries to H_{walk} are identical in the two games and do not contribute.

If $Q = 0$, then $\epsilon = 0$. Otherwise, the reduction selects a uniform padded query slot before the execution and measures its input immediately before that call. If the slot is in the first phase, $\mathcal{D}_1$ stores the outcome for $\mathcal{D}_2$; otherwise $\mathcal{D}_2$ performs the measurement. By [Lemma 6](#), the measured value is F'' with probability at least

$$\Pr[\mathcal{D} \text{ outputs } F''] \geq \frac{W}{Q} \geq \frac{\epsilon^2}{4Q(Q+1)}.$$

Sampling ψ only after the first phase in the actual search game does not affect this probability, since that phase never uses it. The reduction transfers the quantum state once, never rewinds it, and runs at most one prefix of $\mathcal{A}_2$; hence its second-phase depth is $O(d+1) = o(T)$.

Finally, the same finite-family simulation argument as in [Lemma 10](#) and [Appendix A.6](#) can be applied independently to H_{pf} and H_{walk} with sufficiently high-wise independent functions and reversible evaluators. The T programmed walk values are fixed before any adversarial query and kept in an exception table; a canonical enumeration gives the required point-valued outputs, up to negligible

statistical distance. The resulting oracle circuits have polynomial size and cost $O(1)$ rounds per logical query under our round convention. Thus non-negligible ϵ yields a valid depth- $o(T)$ solver for [Assumption 1](#), a contradiction.

B Isogeny-based Classically Secure TLP with Setup

For the sake of completeness, we also present a detailed description of the classically secure TLP with setup based on the conjectured sequentiality of isogeny computations in the $\mathbb{F}_{p^2}$ -isogeny graph, outlined earlier in [Section 2.2](#). This is essentially a simplification of the pairing-based, classically secure TLP with setup based on delay encryption from [\[14\]](#). Compiling this construction using our compiler from [Section 4.2](#) yields a classically secure $\sqrt{T}$ -efficient TLP in the plain model, which was not known from (even classical) isogeny-based assumptions, to the best of our knowledge.

B.1 The Sequentiality Assumption

We first introduce the sequentiality assumption underlying our classically secure TLP with setup. This assumption is based on the conjectured hardness of the *isogeny shortcut problem* that was introduced in [\[20\]](#) (and was used subsequently in [\[17\]](#)). Let $E/\mathbb{F}_{p^2}$ be a supersingular elliptic curve *with unknown endomorphism ring* where $p = O(2^{\text{poly}(\lambda)})$ is a suitably chosen prime such that $(p+1)$ contains a large prime factor $N = \Theta(2^{2\lambda})$ and we fix a small prime ℓ (e.g., a common choice is $\ell = 2$). Over $\mathbb{F}_{p^2}$, the curve E has $(p \pm 1)^2$ points: for simplicity, we restrict ourselves to the case where $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. Throughout this section, it suffices to choose any p and N such that the discrete logarithm (DLOG) problem, and, more specifically, the computational Diffie-Hellman (CDH) problem, is hard over the group of points $E[N]$.

Assumption 2 (Deterministic Multi-Use Isogeny-Shortcut). Let $T = \text{poly}(\lambda)$. Let $p = O(2^{\text{poly}(\lambda)})$ be a suitable prime. Let $N = \Theta(2^{2\lambda})$ be a large prime factor of $p+1$, and let $\ell = O(1)$. Sample a supersingular curve $E/\mathbb{F}_{p^2}$ with unknown endomorphism ring and a seed $\text{seed}_\phi \leftarrow \{0, 1\}^\lambda$. Let $(\phi : E \rightarrow E') \leftarrow \text{DetIsog}(E, \ell, T; \text{seed}_\phi)$, and sample a uniform point $P \in E[N]$ of order N . Set $P' = \phi(P)$. Consider a polynomial-size classical adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. The first phase has total time $\text{poly}(\lambda, T)$. The second phase has parallel depth $o(T)$. The game is as follows.

1. Give $(p, N, \ell, T, E, E', \text{seed}_\phi, \phi, P, P')$ to $\mathcal{A}_1$.
2. The algorithm $\mathcal{A}_1$ outputs a classical state st .
3. Sample $x \leftarrow [0, N-1]$ and set $Q = [x]P$.
4. Give (st, Q) to $\mathcal{A}_2$. It outputs $R \in E'[N]$.

Then $\Pr[R = \phi(Q)] \leq \text{negl}(\lambda)$.

As in [20], we allow the adversary $\mathcal{A}$ a pre-computation taking total time $\text{poly}(\lambda, T)$. Intuitively, this makes the assumption *multi-use*, in the sense that we want it to be computationally intractable for $\mathcal{A}$ to evaluate $\phi(P)$ for uniformly random $P \leftarrow E[N]$, even given an unbounded amount of parallel resources, *and even if* $\mathcal{A}$ is allowed $\text{poly}(\lambda)$ pre-computation after the public parameters $(T, E, E', N, \text{seed}_\phi, \phi)$ are generated (which allows $\mathcal{A}$ to pre-compute $\phi(Y)$ for polynomially many $Y \in E[N]$ of its choice before it sees the challenge point P).

Justification for Assumption 2. For simplicity of exposition, we first focus on the case where the isogeny ϕ is generated via a random walk. We subsequently discuss the case where ϕ is generated via a seeded deterministic walk.

Sequentiality of Isogeny Walk. In the case where the isogeny ϕ is generated via a random walk, any representation of the isogeny ϕ takes at least T bits. So any adversary that reads ϕ sequentially could not evaluate ϕ in time less than T . Thus, any conjecture-breaking adversary needs to parse ϕ in parallel: for any non-redundant representation of ϕ , this seems already hard. The ℓ^T -isogeny ϕ can be decomposed into T many ℓ -isogenies $\phi_1, \dots, \phi_T$, but given the bit-representation of any component ϕ_i , it is hard to compute ϕ_i without knowing its domain, which is given by the codomain of the previous isogeny ϕ_{i-1} . Thus, from a ℓ^T -bit long representation of an ℓ^T -isogeny ϕ , it is unlikely that an adversary can compute the isogeny ϕ in parallel time $o(T)$.

Even if the adversary could somehow compute ϕ , it would also need to evaluate it on a (uniformly random) point P in parallel. This is even harder, and is at the core of the conjecture: mapping a point P under ϕ is equivalent to first mapping it under ϕ_1 , then mapping this output under ϕ_2 , and so on. The input of each step of such a computation depends on the previous one, and is thus inherently hard to parallelize.

Finally, note that isogeny computations are the fundamental building block of essentially all isogeny-based cryptographic protocols. A (partially) parallel algorithm to compute and evaluate isogenies would provide a significant speedup for such protocols: despite the great interest and the practical implications, no such algorithm has been found till date.

The above discussion forms the basis of the conjectured sequentiality of the isogeny shortcut problem, as studied in [20,17,14]. In Assumption 2, we essentially rely on the same conjecture, albeit for a seeded deterministically generated isogeny ϕ . We discuss this next.

Seeded Deterministically Generated Isogeny Walk. The version of the isogeny shortcut problem where the isogeny ϕ is generated via a seeded deterministic walk, while less standard in the literature, plausibly holds because, fundamentally, evaluating an isogeny on a point P is a sequential procedure, and this remains true independently of how the isogeny is sampled.¹ Moreover, even given the

¹ This may not be true in pathological cases: if the isogeny $\phi : E \rightarrow E'$ is sampled in such a way to also produce a short isogeny $\psi : E \rightarrow E'$, the isogeny ϕ may be

public seed seed_ϕ , the definition of the isogeny ϕ is iterative: computing the i -th component ϕ_i requires the curve E_i , while computing E_i requires ϕ_{i-1} and E_{i-1} . It thus seems implausible that an adversary could compute ϕ and evaluate it on a point in parallel time less than T . In [Assumption 2](#), we use a seeded deterministically generated isogeny ϕ , while relying on the hardness of the corresponding adaptation of the isogeny shortcut problem.

Comparing Isogeny Shortcut and [Assumption 2](#). [Assumption 2](#) fundamentally relies on two hardness assumptions: (i) the hardness of the CDH problem over $E[N]$ (which in turn implies the hardness of the DLOG problem over $E[N]$), and (ii) the sequential hardness of the isogeny shortcut problem [20,17] for deterministically represented isogeny walks (as discussed above).

- The hardness of DLOG guarantees that a poly-bounded adversary cannot find a shortcut to the solution $Q' = \phi(Q)$ by first computing the discrete log x such that $Q = [x]P$ and then obtaining Q' as $Q' = [x]P'$, thus immediately leading to a distinguisher. More concretely, we rely on the CDH assumption for a stronger guarantee, namely, that a poly-bounded adversary, given (P, P', Q) cannot distinguish Q' from a uniformly random point in $E'[N]$.
- Assuming the hardness of CDH (and hence the hardness of DLOG),¹ [Assumption 2](#) hinges on the fact that the only other way to obtain Q' is by computing $Q' = \phi(Q)$, which—by the sequential nature of a ℓ^T -isogeny in the $\mathbb{F}_{p^2}$ -graph requires at least parallel time T . This is precisely the isogeny shortcut problem, as studied in [20,17,14], for seeded deterministically generated isogeny walks (as discussed above, this is also plausibly sequential).

In other words, the main distinction between the (seeded deterministic) isogeny shortcut assumption from prior works [20,17] (for a seeded deterministically generated isogeny walk) and [Assumption 2](#) is that the adversary $\mathcal{A}$ in [Assumption 2](#) is given, in addition to the public parameters $(p, N, \ell, T, E, E', \text{seed}_\phi, \phi)$, a pair of points $(P, P' = \phi(P)) \in E[N] \times E'[N]$. However, observe that the adversary is allowed $\text{poly}(\lambda, T)$ pre-computation, which allows it to potentially sample multiple pairs of the form $(P, \phi(P))$. So the additional presence of $(P, \phi(P))$ in the public parameters given to the adversary is not a strengthening of the isogeny shortcut problem in any way.

Finally, we remark that the attacks against SIDH [15,26,31] and the techniques that rely on higher-dimensional isogenies cannot help break [Assumption 2](#). Even if the action of the isogeny ϕ on the point P is given, extracting a higher-dimensional representation of ϕ would require computing an isogeny (in higher dimensions) of degree $N = \Theta(2^{2\lambda})$, which is a large prime. Thus, any possible application of higher-dimensional representation techniques requires computations whose complexity is exponential, and thus much larger than T .

evaluated efficiently by evaluating ψ instead. This is clearly not the case in our construction, where the isogeny ϕ is obtained from a uniform seed by [Algorithm 1](#).

¹ The adversary has access to $\text{poly}(\lambda)$ many parallel processors: thus, if the DLOG/CDH problem is hard, it cannot be solved in parallel time $T = \text{poly}(\lambda)$.

B.2 Construction of Classically Secure TLP with Setup

In this subsection, we present the construction of classically secure TLP with setup based on [Assumption 2](#). We assume an efficient algorithm `GenCurve`¹ that takes as input a security parameter λ and outputs a tuple of the form (p, N, ℓ, E, P) as in [Assumption 2](#) in time $O(\text{poly}(\lambda))$. Let $H : E'[N] \rightarrow \{0, 1\}^\lambda$ be a hash function, modeled as a random oracle. Note that such a hash function can be realized by first applying a canonical encoding $\text{enc} : E'[N] \rightarrow \{0, 1\}^n$ for $n = \text{poly}(\lambda)$, followed by a standard hash function that maps n -bit strings to λ -bit strings.

PSetup $(1^\lambda, T)$: Given a security parameter λ and a sequentiality parameter T , the setup algorithm proceeds as follows:

1. Sample $(p, N, \ell, E, P) \leftarrow \text{GenCurve}(1^\lambda)$.
2. Sample $\text{seed}_\phi \leftarrow \{0, 1\}^\lambda$ and compute the ℓ^T -isogeny $\phi : E \rightarrow E'$ as the output of $\text{DetIsog}(E, \ell, T; \text{seed}_\phi)$ ([Algorithm 1](#)).
3. Compute $P' = \phi(P)$.
4. Output $\text{pp} = (p, N, \ell, T, E, E', \text{seed}_\phi, P, P')$.

PGen $(\text{pp}, s \in \{0, 1\}^\lambda)$: Given $\text{pp} = (p, N, \ell, T, E, E', \text{seed}_\phi, P, P')$ and a solution $s \in \{0, 1\}^\lambda$, the puzzle generation algorithm proceeds as follows:

1. Sample $x \leftarrow [0, N - 1]$.
2. Compute $Q = [x]P \in E[N]$ and $Q' = [x]P' \in E'[N]$.
3. Compute $e = H(Q') \oplus s$.
4. Output the puzzle instance $Z = (\text{pp}, Q, e)$.

PSolve (Z) : Parse $Z = (\text{pp}, Q, e)$, where $\text{pp} = (p, N, \ell, T, E, E', \text{seed}_\phi, P, P')$. Then do the following:

1. Compute the ℓ^T -isogeny $\phi : E \rightarrow E'$ as the output of $\text{DetIsog}(E, \ell, T; \text{seed}_\phi)$ ([Algorithm 1](#)).
2. Compute $Q'' = \phi(Q)$.
3. Output $s' = e \oplus H(Q'')$.

B.3 Efficiency Analysis

In this subsection, we present an analysis of the efficiency of the above TLP with setup construction.

¹ The algorithm `GenCurve` is efficient: finding suitable p , N , and ℓ can be done efficiently, for instance, by setting $\ell = 2$, N as the smallest prime larger than $2^{2\lambda}$, and p as the smallest prime such that $N \mid p + 1$. The curve E is chosen by a long random walk in the isogeny graph, and the point P can be sampled efficiently given E (see [Section 3.3](#)).

Parameters. Let λ be the security parameter and T be the sequentiality parameter. Note again that in general, the sequentiality parameter T is a (large) polynomial in λ . For some applications, one may consider T as super-polynomial in λ . For the TLP construction above, we set:

$$p = O(2^{\text{poly}(\lambda)}), N = \Theta(2^{2\lambda}), \ell = O(1).$$

for large enough p, N such that the CDH (and hence, DLOG) problem is hard over $E[N]$. For a practical instantiation, we recommend setting $\ell = 2$. Note that all of these parameters are independent of the sequentiality parameter T , which sets the length of the seeded deterministic non-backtracking walk generated by DetIsog .

Efficiency of Setup. First we argue that PSetup takes time $O(T \cdot \text{poly}(\lambda))$ for the choice of parameters described above. To see this, observe that the setup algorithm computes an isogeny of degree ℓ^T , which takes time $O(T \cdot \text{poly}(\lambda))$ and space $O(T \cdot \text{poly}(\lambda))$ bits to represent. Further, since $p = O(2^{c\lambda})$, $N = \Theta(2^{2\lambda})$ for $c = O(1)$ and $\ell = O(1)$, all of the other computations performed by the setup take time $O(\text{poly}(\lambda))$. Hence, overall, setup takes time $O(T \cdot \text{poly}(\lambda))$.

Efficiency of Puzzle Generation. Next we argue that PGen takes time $O(\lambda)$ for the choice of parameters described above. This follows from the fact that $\text{ord}(P) = \text{ord}(P') = N = \Theta(2^{2\lambda})$, and hence, both $Q = [x]P \in E$ and $Q' = [x]P' \in E'$ for any $x \in [0, N-1]$ can be computed in time $O(\lambda)$ using the usual square-and-multiply algorithm given the $O(\log N) = O(\lambda)$ -bit representation of x . All additional group/point operations in PGen also take time $O(\log N) = O(\lambda)$ via standard elliptic curve operations. In particular, note that puzzle generation time is independent of T .

Efficiency of Puzzle Solving. The solver reconstructs $\phi = \text{DetIsog}(E, \ell, T; \text{seed}_\phi)$ from the public curve E , delay T , and seed seed_ϕ . It evaluates the seeded deterministic walk of length T on Q . This computation takes $O(T \cdot \text{poly}(\lambda))$ time.

B.4 Correctness Analysis

We state and prove the following theorem for the correctness of our construction of TLP with setup.

Theorem 7 (Correctness of Our Classical TLP with Setup). *For any security parameter λ , any sequentiality parameter T , and any solution $s \in \{0, 1\}^\lambda$, letting $\text{pp} \leftarrow \text{PSetup}(1^\lambda, T)$ and $Z \leftarrow \text{PGen}(\text{pp}, s)$, we have $\text{PSolve}(Z) = s$.*

Proof. Note that, by construction, we have $Z = (\text{pp}, [x]P, H([x]P') \oplus s)$ for some $x \in [0, N-1]$, where $P' = \phi(P)$. Next, observe that, in the computation of $\text{PSolve}(Z)$, the puzzle solver computes $Q'' = \phi(Q)$ by performing the non-backtracking walk of length T determined by ϕ in the ℓ -isogeny $\mathbb{F}_{p^2}$. Hence, we have

$$Q'' = \phi(Q) = \phi([x]P) = [x] (\phi(P)) = [x]P'$$

where the third equality follows from the homomorphic property of the isogeny map ϕ . Hence, we have

$$\begin{aligned}\text{PSolve}(Z) &= H(Q'') \oplus (H([x]P') \oplus s) \\ &= H([x]P') \oplus (H([x]P') \oplus s) \\ &= s.\end{aligned}$$

This completes the proof of correctness.

B.5 Security Analysis

We state and prove the following theorem, which captures the security of our classically secure TLP construction.

Theorem 8 (Security of Our Classical TLP with Setup). *Under [Assumption 2](#) and assuming that H is modeled as a random oracle, the above construction of TLP with setup satisfies reusable security as per [Definition 4](#).*

Proof. Suppose that a polynomial-size classical adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ breaks reusable security with advantage ϵ , where $\mathcal{A}_1$ runs in time $\text{poly}(\lambda, T)$ and $\mathcal{A}_2$ has parallel depth $o(T)$. Let $q_H = \text{poly}(\lambda)$ bound the total number of random-oracle queries made by $\mathcal{A}$ across both phases. We construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against [Assumption 2](#) as follows:

1. The algorithm $\mathcal{B}_1$ receives as input a tuple of the form $(p, N, \ell, T, E, E', \text{seed}_\phi, \phi, P, P')$ and gives

$$\text{pp} = (p, N, \ell, T, E, E', \text{seed}_\phi, P, P')$$

to $\mathcal{A}_1$. It simulates the random oracle $H : E'[N] \rightarrow \{0, 1\}^\lambda$ by lazy sampling and records in a list L every input queried to H . When $\mathcal{A}_1$ outputs (z, s_0, s_1) , the reduction passes (z, s_0, s_1, pp) , together with the current random-oracle table and the list L , to the second phase algorithm $\mathcal{B}_2$.

2. The algorithm $\mathcal{B}_2$ receives the fresh challenge point $Q = [x]P$ from the challenger in the security game for [Assumption 2](#). It samples independent and uniformly random $b \leftarrow 0, 1$ and $u \leftarrow 0, 1^\lambda$, forms

$$Z = (\text{pp}, Q, u \oplus s_b),$$

and runs $\mathcal{A}_2(Z, z)$ while continuing the same lazy-sampled random oracle strategy and recording its queries in L . After $\mathcal{A}_2$ terminates, $\mathcal{B}_2$ outputs a uniformly chosen element of L (or an arbitrary fixed point of $E'[N]$ if $L = \emptyset$).

It remains to analyze the reduction. Let $Q^* = \phi(Q)$. In the above simulated game, u is independent of b , and hence $\mathcal{A}_2$ guesses b with probability exactly $1/2$. In the real TLP experiment, the mask is instead $H(Q^*)$. Couple the two experiments by programming $H(Q^*) = u$ whenever Q^* has not previously been

queried. Until the adversary queries H at Q^* , the two experiments are identically distributed. Therefore, $\epsilon \leq \Pr[Q^* \in L]$. Consequently, for $q_H > 0$,

$$\Pr[\mathcal{B}_2 \text{ outputs } Q^*] \geq \frac{\epsilon}{q_H},$$

while if $q_H = 0$ then necessarily $\epsilon = 0$. Since q_H is polynomial, a non-negligible ϵ gives a non-negligible probability of outputting $\phi(Q)$.

Finally, observe that $\mathcal{B}_1$ runs in time $\text{poly}(\lambda, T)$, while $\mathcal{B}_2$ invokes $\mathcal{A}_2$ only once and performs only polynomially bounded local work. Thus its parallel depth remains $o(T)$. This contradicts [Assumption 2](#) and proves reusable security for our classical TLP with setup, as desired.