

“Preview Writeup”: In anticipation of a package submission to the NIST Threshold Call

Title: Axolotl

Subtitle: Robust Asynchronous Two-Round Threshold Schnorr over Point-to-Point Channels

Version: Preview Writeup 1.1 (2026-09-24)

Team name: Team Trout

Team members: Hila Dahari-Garbian, Ariel Nof, Luke Parker

Abstract: Our submission will specify Axolotl, a gadget (category S7) which enables verifiable evaluation of a pseudo-random function *in the exponent* by a set of parties in just one round. Axolotl is robust, having each participant simply multi-cast a single message of size *constant to the security parameter* to invoke. Each message is less than 2 kibibytes for the desired security parameters.

We then use Axolotl to instantiate a threshold Schnorr signing functionality (category N1.1). The first round of our protocol is used to generate the nonce commitment. The second round of our protocol is used to output the response to the challenge. By re-evaluating the Axolotl gadget to sample the nonce with each round, we manage to avoid needing to deal shares for the nonce across rounds and maintain a transcript of size linear in the set size, effecting linear per-participant communication. This also effects a stateless protocol in that it requires *no* storage other than the setup and a set of *any* messages from the first round of size exceeding the threshold.

Our signing protocol supports any participation threshold $t \leq n$, where n is the amount of parties, and the optimal corruption threshold of $t - 1$ corruptions. It is interoperable with existing Shamir-secret-shared signing keys.

Proposed crypto-systems: (I) Axolotl (Categories N1.1, S7).

Keywords: Threshold Cryptography; NIST Threshold Call; Schnorr; Class Groups

Preview writeup. This document is provided to NIST for online publication, to foster public awareness and support public discussion within the scope of the NIST First Call for Multi-Party Threshold Schemes [NIST-IR8214C]. This “preview writeup” represents a good-faith plan for a subsequent “package submission”. However, until the deadline for package submission, the team may still modify its own composition and the submission plan, including possible changes to the technical scope, and/or the used techniques or achieved results.

Team members: Hila Dahari-Garbian^{i1,a1,*}, Ariel Nof^{i2,a2,*}, Luke Parker^{i3,a3}

Open Researcher and Contributor Identifiers (ORCID):

i1 (0009-0002-0122-552X); i2 (0009-0007-4852-083X); i3 (0009-0008-0432-3655)

Affiliations:

^{a1} Efi Arazi School of Computer Science, Reichman University @ Herzliya, Israel

^{a2} Department of Computer Science, Bar-Ilan University @ Ramat-Gan, Israel

^{a3} Serai DEX @ Florida, United States of America

Associateship clarifications:

* Ph.D.

Main contacts:

- **Team mailing list:** teamtrout@googlegroups.com
- **Primary technical contact person:** Luke Parker, lukeparker@serai.exchange
- **Secondary contact person 1:** Ariel Nof, ariel.nof@biu.ac.il
- **Secondary contact person 2:** Hila Dahari-Garbian, hiladahari41@gmail.com

Produced by humans. The team hereby confirms that the content in this preview writeup: (i) was produced by the team members, and (ii) was not produced by generative artificial intelligence (AI), with the possible exception of AI-proposed grammar improvements, minor integrated suggestions, or some well-identified and short localized portions of auxiliary content (e.g., some illustration); and (iii) was proofread by the team members.

1. Introduction

The notion of an exponent-Verifiable Random Function (eVRF) [B HLS25] is a pseudo-random function evaluated with a key as input, its output committed to in the exponent of a group where the (elliptic-curve) discrete-logarithm problem is hard, yet still enabling efficient verification that the function was invoked correctly. When proposed, an open question was asked for an efficient construction of a key-homomorphic eVRF. A key-homomorphic eVRF would be such that the interpolation of the evaluations from a set of keys was congruent to the evaluation for a corresponding interpolation of the keys, as would easily allow thresholdizing the eVRF. While we do not present a key-homomorphic eVRF, we present what we consider a multi-party eVRF, where there exists a multi-party protocol to evaluate an eVRF. We find this protocol sufficient for most desired use-cases for a key-homomorphic eVRF, and have devised an efficient practical construction.

We name our result Axolotl, referring to the axolotls' ability to regrow limbs. This embodies the spirit of our work, where sets of participants are only decided *at the end of each round*, and the result of each round is *independent of the set which participated*. This allows us to lose a participant from the first round of our Schnorr signing protocol, so long as we gain any other participant in the second round, and allows us to avoid requiring agreement on the set of participants. Not only does this result in a much simpler and more efficient protocol than those which require consistent sets of participants, working even over asynchronous networks, but also supports *arbitrary thresholds*, including the dishonest majority case.

We construct the MP-eVRF from an unknown order group with a known-order subgroup where the discrete-log problem is easy. We note a construction over Paillier [Pai99] is possible, but ourselves use class groups as seen within Castagnos-Laguillaumie [CL15] for the feasible transparent setup and ability to choose our message-space. At a high level, the design is to sample a uniform group element where the discrete-logarithm for the term from the known-order subgroup serves as our pseudo-random value. We design a one-round threshold protocol to perform a masked reveal of this term, maintaining its privacy while allowing us to create commitments for and derivatives of it.

We then use Axolotl to instantiate a threshold Schnorr signing functionality, the pseudo-random function being used to derive the signatures' nonces, with verification equivalent to EdDSA. While our nonces are *not* derived as specified in FIPS-186-5 [Nat23], they are deterministic to the message being signed and independent of the participants used to invoke the signing protocol.

We believe our signing protocol is exceptionally clear to deploy, not requiring composition with *any* other protocols or functionalities to be deployed in a real-world environment. Despite this simplicity, it maintains *linear* per-participant communication complexity, is round optimal, and still maintains concrete efficiency at less than 4 kibibytes multi-cast per participant for 128-bit security.

Our work has yet to be published but we expect to publish it via an open forum in a week or so.

2. Specification

- **Organization:**

The specification will include references to existing literature as relevant to the framework underlying our protocol. We do not plan to restate its complete history, but will include all definitions necessary to prove the correctness and security of Axolotl and the resulting functionality for Schnorr/EdDSA. We will include an academic description as traditional for papers (and as the reference for the included security proofs) before continuing with an exact technical specification. In doing so, we hope to clearly present the theory to the people who will review the functionality, but also the practicals to those people who will actually implement it.

The Axolotl gadget is not limited to class groups (as seen within [CL15]) but our submission will specify Axolotl instantiated over such class groups. Team Trout is actively working on a distinct submission, Trout++, whose preview [DNP26] already discussed how it will cover class groups in exceptional detail. We plan to use the same background for this submission, as applicable, and maintain the same level of detail overall. Due to this, this section is largely a restatement of the Trout++ preview's section on specification.

One popular reference for learning, and programming, binary quadratic forms is Henri Cohen's A Course in Computational Algebraic Number Theory [Coh00]. It may be incredibly appreciated for including descriptions of algorithms in English plain enough for most developers, even those without a background in the relevant mathematics, to implement binary quadratic forms using just a library such as the GNU Multiple Precision Arithmetic Library (GMP). We note BICYCL [BCIL23], the leading implementation for binary quadratic forms, internally makes use of GMP for its rich set of functions and performance.

We plan to go a step further by describing algorithms accompanied with a Python-esque pseudo-code, avoiding the sole use of English descriptions for our functionality (and its required methods). Each method will be well-defined (e.g. one function for the composition of two binary quadratic forms) with clarity on the inputs, outputs, and associated bounds. While we will provide an implementation we expect to be portable and clear, we will also cite alternative methods known to be equivalent, possibly offering greater performance at the cost of increased implementation effort. This will provide an accurate compilation of the field while keeping the difficulty of implementation low.

Additionally, we plan to present the bounds on each *intermediary value* within an algorithm. While most algorithms prove correctness and a bound for termination, they frequently omit the exact properties of each step. In practice, this requires the developer to derive bounds themselves (placing the developer in the role of the mathematician), or requires the developer to use unbounded operations (which are less efficient and with undefined memory consumption). We plan this thoroughness with the explicit intent of enabling *non-allocating*, *constant-time* implementations, as present as an option within Trout++'s reference implementation. Non-allocating implementations benefit from having *completely static* memory usage, removing the potential for allocation failures at runtime, and enabling explicitly boxing any implementation with the exact memory it will need. Constant-time

implementations benefit from being resilient to side-channel attacks premised on timing information, a leading source of leakage when the attacker only has remote access to the signer.

Our work on the implementation of class groups, as seen within our framework of choice and derivatives ([CL15], [CLT18]), will be of independent interest to any candidate which shares the same framework. Similarly, the work on BICYCL (for which three of the paper's four authors are members of the team "BICYCLIST" which submitted previews in the first round) regarding both the algorithmic design and literal implementation of binary quadratic forms will be of benefit to our submission and reference implementation.

- **System Model:**

Axolotl requires a integer key as the input to its underlying pseudo-random function. For this, we derive a setup from [BDO23], assuming a synchronous network for all participants. For the signing key, our threshold Schnorr protocol is explicitly defined to work with existing Shamir-secret sharings. This decision naturally allows it to compose with any Distributed Key Generation functionality and even allows adoption with pre-existing keys.

The threshold Schnorr protocol is best considered as the composition of two one-round protocols. The first protocol, corresponding to the first round, invokes the Axolotl gadget once to learn the commitment to the nonce. Anyone who receives t messages from honest parties (each of size constant in the amount of parties) may learn the nonce's commitment, while malicious participants can be identified via the included zero-knowledge proofs. The nonce is pseudo-random to the setup and *independent of the set used to reconstruct it*. This allows us to circumvent the honest majority bounds traditionally required for agreement, and the complexity of agreement, by circumventing the need for agreement entirely. While traditionally, this would imply techniques such as Fully Homomorphic Encryption, we are able to instantiate Axolotl at approximately the cost of a single exponentiation in an unknown-order group (and associated proof of correctness).

The second protocol, corresponding to the second round, simply invokes the Axolotl gadget to derive the nonce again. While in the first round, the nonce is published *in the exponent* to form the nonce commitment, in the second round, the nonce is summed with the signing key according to the linear derivation which forms the response to the challenge. This again solely requires receiving t messages from honest parties, each of size constant in the amount of parties, where malicious participants may be identified via the included zero-knowledge proofs.

The Axolotl gadget and corresponding Schnorr protocol are proven with simulation-based proofs. For n participants and a participation threshold t signers required to produce a signature, we support any $t \leq n$ and soundness against $t - 1$ static corruptions. When the amount of corruptions is greater than $n - t$, the adversary may indefinitely stall (but not explicitly abort) the protocol by simply omitting their messages, but the protocol maintains liveness so long as t honest signers are able to communicate. We limit our assumptions and prefer more conservative choices for zero-knowledge proofs (explaining why we may have somewhat higher bandwidth than other schemes premised on class groups). To that end,

we only require the Decisional Diffie-Hellman assumption [CL15] and the Adaptive Root Assumption [Wes19].

3. Open-Source Implementation

The current reference implementation is a group of libraries written in the Rust programming language. Our first library defines an interface for working with binary quadratic forms, and common methods for implementations, yet allows the user to choose the specific implementation. This allows, when a C++ compiler is provided and code licensed under the GNU General Public License version 3 is allowed, to use the performant BICYCL library. Alternatively, a pure-Rust implementation whose methods operate in time independent of their inputs may be used (avoiding leaking timing information about the witness, as used for side-channel attacks).

The current reference implementation also makes use of Rust libraries from the Rust ecosystem, colloquially referred to as "crates", as managed via the cargo package manager. The exact libraries are amenably licensed and may be vendored into a single source tree as preferable. We believe we use a minimal and reasonable set of libraries, given the ecosystem, yet still actively work on minimization to ensure the implementation's independence, ability to be reviewed, and security against supply-chain attacks. We plan to use cargo to build our implementation and run tests, it being the de facto tool in the Rust ecosystem.

The current reference implementation is still a work in progress. Other works of ours, which are being used for this submission as well, have a publicly available Git repository available via <https://github.com/kayabaNerve/trout> however.

4. Experimental Performance Evaluation

Our experimental implementation had each participant multi-cast less than 2 KiB per invocation of the Axolotl gadget. The computational cost is effectively solely the computation of a Diffie-Hellman over the class group, with the associated zero-knowledge proof. We are not ready to confirm exact computational performance numbers at this time, but expect them to be within a few hundred milliseconds in variable-time, or in a few seconds in constant-time, on a single thread.

For the computational performance, we hope to improve with a more efficient implementation as we prepare our final submission. We are currently aware of some methods for which we currently do not implement the state of the art and, as time allows, can tighten some worst-case bounds to reduce the amount of iterations algorithms perform when run in constant time.

We do not expect challenges running on the baseline platform due to our single-thread-per-party implementation, without target-specific optimizations other than those the compiler automatically applies. While multiple threads would benefit proving, we should be able to accept any limitations employed by the baseline platform (which needs to run each participant's view) and provide our measurements/reasoning if necessary.

5. Licensing, Patent Claims, and Funding

We do not know of any patents that do or could have claims that may cover the contents of our submission. We attest that no member of our team has applied for any patents in any jurisdiction for any part of our submission. The current reference implementation is being developed under the GNU Affero General Public License Version 3.0 (AGPL-3.0-only) yet may also be made available under a less restrictive OSI-approved license (yet to be decided) and currently has (possible) dependencies under the following licenses (listed by their SPDX identifiers).

- BSD-2-Clause
- BSD-3-Clause
- MIT
- Zlib
- Apache-2.0
- LGPL-3.0-only
- LGPL-3.0-or-later
- Unicode-3.0

All are OSI Approved Licenses, recognized to comply with the Open Source Definition.

6. Comparison with Other Submissions

We are unaware of any other robust threshold Schnorr protocols submitted at this time. While FROST [KG20] may be made robust via ROAST [RRJS22], the resulting protocol has total communication complexity of $\mathcal{O}(n^4)$ (compared to our $\mathcal{O}(n^2)$). We are also unaware of any other asynchronous threshold Schnorr signing protocols submitted at this time, despite real networks not being ideal (a key consideration when discussing deployment and standardization).

FROST is proven secure against adaptive adversaries for certain thresholds, or under the novel Low-Dimensional Vector Representation (LDVR) assumption [CKTZ25]. (Red)ETA's Schnorr protocol [BBLMPS26] is proven secure against adaptive adversaries when the adversary does not participate in key generation. FaFROST [BGCRSY26], Gargos [BDLR26], and Classic Schnorr [AABCSGM26] are proven against adaptive adversaries. Our protocol allows identifying misbehaving participants (though they are unable to even abort the protocol) with the messages they send, like FROST, while FaFROST, Gargos, and Classic Schnorr require extensions to achieve identifiable aborts. (Red)ETA does not specify if they allow identifying misbehaving participants (and if so, how).

Our Schnorr protocol completes in two rounds, like FROST and FaFROST, while Gargos and Classic Schnorr require three rounds. Our per-participant communication is $\mathcal{O}(n)$, the same as FROST, FaFROST, and Gargos, with the caveat that FROST and FaFROST assume a coordinator acts as a bulletin board and the coordinator seemingly incurs $\mathcal{O}(n^2)$ communication per execution.

Classic Schnorr differs by explicitly requiring a synchronous *broadcast* channel which may increase the round complexity in practice. We assume Classic Schnorr to maintain $\mathcal{O}(n)$ per-participant communication however. (Red)ETA has been excluded from this comparison due to not publishing its complexities within their preview.

Finally, we are unaware of any *deterministic* threshold Schnorr signing protocols submitted at this time other than our own. While (Red)ETA's Schnorr protocol does describe deterministic nonces via the Purify pseudo-random function, this presumably only makes *shares* of the nonce deterministic, with the resulting nonce still variable to the exact set of participants used to sample the nonce.

References

- [AABCSGM26] Michael Adedj, Tomer Ashur, Amit Singh Bhati, Geoffroy Couteau, Cyprien Delpech de Saint Guilhem, Michael Gutkin, and Nikos Makriyannis. *Distributed Schnorr Signatures*. 2026. URL: <https://csrc.nist.gov/csrc/media/Projects/threshold-cryptography/documents/TCall-1/Fireblocks-3MI-a-PW01.pdf>.
- [BBLMPS26] Alessandro Barengi, Michele Battagliola, Riccardo Longo, Alessio Meneghetti, Gerardo Pelosi, and Edoardo Signorini. *(Red)ETA: Refreshable Extensible DLOG Enhanced Threshold Algorithms*. 2026. URL: <https://csrc.nist.gov/csrc/media/Projects/threshold-cryptography/documents/TCall-1/RedETA-PW01.pdf>.
- [BCIL23] Cyril Bouvier, Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. “I want to ride my BICYCL: BICYCL Implements CryptographY in CClass groups”. In: *Journal of Cryptology* 36 (2023). DOI: [10.1007/s00145-023-09459-1](https://doi.org/10.1007/s00145-023-09459-1). Also at ia.cr/2022/1466.
- [BDLR26] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. *Gargos: Threshold Schnorr Signature Scheme*. 2026. URL: <https://csrc.nist.gov/csrc/media/Projects/threshold-cryptography/documents/TCall-1/BDLR-Gargos-PW01.pdf>.
- [BDO23] Lennart Braun, Ivan Damgård, and Claudio Orlandi. “Secure Multiparty Computation from Threshold Encryption Based on Class Groups”. In: *Advances in Cryptology - CRYPTO*. 2023, pp. 613–645. DOI: [10.1007/978-3-031-38557-5_20](https://doi.org/10.1007/978-3-031-38557-5_20). Also at ia.cr/2022/1437.
- [BGCRSY26] Ruben Baecker, Paul Gerhart, Davide Li Calsi, Luigi Russo, Dominique Schröder, and Arkady Yerukhimovich. *A FROST-based Threshold Schnorr Signature Scheme with Identifiable Aborts and Fully Adaptive Security from AOMDL*. 2026. URL: <https://csrc.nist.gov/csrc/media/Projects/threshold-cryptography/documents/TCall-1/FaFrost-PW02.pdf>.
- [BHLS25] Dan Boneh, Iftach Haitner, Yehuda Lindell, and Gil Segev. “Exponent-VRFs and Their Applications”. In: *Advances in Cryptology - EUROCRYPT*. 2025, pp. 195–224. DOI: [10.1007/978-3-031-91098-2_8](https://doi.org/10.1007/978-3-031-91098-2_8). Also at ia.cr/2024/397.
- [CKKTZ25] Elizabeth Crites, Jonathan Katz, Chelsea Komlo, Stefano Tessaro, and Chenzhi Zhu. “On the Adaptive Security of FROST”. In: 2025. DOI: [10.1007/978-3-032-01887-8_16](https://doi.org/10.1007/978-3-032-01887-8_16). Also at ia.cr/2025/1061.
- [CL15] Guilhem Castagnos and Fabien Laguillaumie. “Linearly Homomorphic Encryption from DDH”. In: *Topics in Cryptology - CT-RSA*. 2015, pp. 487–505. DOI: [10.1007/978-3-319-16715-2_26](https://doi.org/10.1007/978-3-319-16715-2_26). Also at ia.cr/2015/047.
- [CLT18] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. “Practical Fully Secure Unrestricted Inner Product Functional Encryption Modulo p ”. In: *ASIACRYPT*. 2018, pp. 733–764. DOI: [10.1007/978-3-030-03329-3_25](https://doi.org/10.1007/978-3-030-03329-3_25). Also at ia.cr/2018/791.

- [Coh00]** Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 2000. DOI: [10.1007/978-3-662-02945-9](https://doi.org/10.1007/978-3-662-02945-9).
- [DNP26]** Hila Dahari-Garbian, Ariel Nof, and Luke Parker. *Trout++: Two-Round Threshold ECDSA Composable with ROAST*. 2026. URL: <https://csrc.nist.gov/csrc/media/Projects/threshold-cryptography/documents/TCall-1/Trout-PW02.pdf>.
- [KG20]** Chelsea Komlo and Ian Goldberg. “FROST: Flexible Round-Optimized Schnorr Threshold Signatures”. In: *Selected Areas in Cryptography - SAC*. 2020, pp. 34–65. DOI: [10.1007/978-3-030-81652-0_2](https://doi.org/10.1007/978-3-030-81652-0_2). Also at ia.cr/2020/852.
- [Nat23]** National Institute of Standards and Technology. *Digital Signature Standard (DSS)*. 2023. DOI: [10.6028/NIST.FIPS.186-5](https://doi.org/10.6028/NIST.FIPS.186-5). URL: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf>.
- [Pai99]** Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology - EUROCRYPT 1999*. 1999. DOI: [10.1007/3-540-48910-X_16](https://doi.org/10.1007/3-540-48910-X_16).
- [RRJSS22]** Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. “ROAST: Robust Asynchronous Schnorr Threshold Signatures”. In: *ACM CCS*. 2022, pp. 2551–2564. DOI: [10.1145/3548606.3560583](https://doi.org/10.1145/3548606.3560583). Also at ia.cr/2022/550.
- [Wes19]** Benjamin Wesolowski. “Efficient verifiable delay functions”. In: *EUROCRYPT*. 2019. DOI: [10.1007/978-3-030-17659-4_13](https://doi.org/10.1007/978-3-030-17659-4_13). Also at ia.cr/2018/623.
- [NIST-IR8214C]** Luís T. A. N. Brandão and René Peralta. *NIST First Call for Multi-Party Threshold Schemes*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8214C. 2026. DOI: [10.6028/NIST.IR.8214C](https://doi.org/10.6028/NIST.IR.8214C).