Firn Protocol | Technical Whitepaper

Captain McAteer | firnprotocol@proton.me

Firn is an advanced, zero-knowledge privacy platform for Ethereum and EVM-based L2s, which overcomes certain key technical obstacles that competing services face. Specifically, Firn admits a highly lightweight, client-side *wallet*, which runs exclusively in the web browser, and efficiently retrieves its user's account balance and state (and moreover generates zero-knowledge proofs on this user's behalf). Other protocols impose significant computational and bandwidth complexity burdens on their users' wallets. That is, competing privacy protocols—including *Aztec Connect* and *Tornado Cash Nova*, as well as standalone chains like *Zcash* and *Monero*—all impose *linear* computational and bandwidth burdens on their wallets, in the sense that each user, upon logging in, must perform computational work proportional to the *time elapsed* since the user's last synchronization. This requirement induces synchronization times in the minutes, and download requirements in the megabytes, even for regular users. This problem has proven prohibitive in multiple real-life instances. Aztec Connect recently shut down, for example—plausibly, because of runaway reports of an unusable service riddled with bugs, and of inaccessible funds—and Zcash, simultaneously, has been beset by a severe "spam attack", which has made many wallets unusable.

Efficiency by Design

Firn bypasses this endemic problem entirely, by adopting an *account-based* architecture. *All prior* privacy services—that is, every extant service except for Firn—uses a *UTXO-based* architecture, in which funds reside in Bitcoin-like "notes", and are redeemed in a privacy-preserving way. That is, these protocols operate essentially like Bitcoin does, except that they cryptographically hide the numeric *values* stored in their UTXOs, and moreover conceal *which* UTXO, in any given transaction, is actually being consumed.

This model makes it computationally difficult to *receive* funds, as we now explain. In order to learn its own account state, each given wallet—in a UTXO-based system—must download, and trial-decrypt, *every* output TXO posted to the network since the wallet last synchronized. Moreover, it must remain online as frequently as possible, as well as cache state on its device. This makes synchronization even more costly on fresh devices; in any case, it also presents serious security risks (that is, those associated with caching state on possibly-untrusted devices). In short, in UTXO-based systems, each given user's funds may be, in general, spread throughout hundreds of UTXOs, and these UTXOs must be found.

In the setting of Bitcoin, wallets manage this task—i.e., that of state management—with the aid of powerful, external indexing servers. This recourse is not available in the cryptographically *private* setting, for obvious reasons; indeed, to leverage an external server, a wallet would have to hand over to that server its user's "viewing key", thereby *permanently and unrevocably* giving that server viewing access into the user's activity (past and future). (In fact, an early Zcash wallet, amazingly, employed precisely this measure!) These protocols thus force their wallets to bear this burden alone, with consequences already made clear above (or else severely compromise their users' security).

Firn's architecture, on the other hand, is account-based, as we now explain. Just as Ethereum, unlike Bitcoin, stores its users' states in a global table of *accounts*, Firn, unlike UTXO-based privacy protocols, likewise stores its users' account states in a global table, in which these users' account balances, now, remain continually *encrypted*. Crucially, when each given user *receives* a payment, the user's *new* account balance ciphertext overwrites its *old* one. As a consequence, a user could, say, receive hundreds of payments, over the course of a year—and remain *offline* throughout that entire duration, no less—and, upon logging back in, nonetheless recover his recent account state instantly.

Our thesis is that this feature will prove—and, in fact, is already proving—to be decisive in the race between privacy protocols. Firn's client is extremely lightweight, and must download *less than a kilobyte* of data in order to retrieve its user's account state (regardless of how long the user has been offline); moreover, Firn's *never* caches state on the device, and is lightweight enough to easily run in Tor and on mobile devices.

Improving State-of-the-Art Protocols

Firn is based on a sequence of academic works, called *Zether* [1] and *Anonymous Zether* [2], which introduce the account-based privacy model. Firn builds upon these works, by implementing various theoretical and practical improvements, as well as filling certain gaps. For example, we implement all *deposits*, *transfers*, and *withdrawals* in a privacy-preserving manner; [2] only discusses private transfers (the generalization to deposits and withdrawals is straightforward). More significantly, using a few new ideas, we are able to reduce the *verifier's complexity* of Anonymous Zether to $O(N)$, in the anonymity set size N, as opposed to $O(N \log N)$. We refer to the paper [2] in what follows, and use that paper's notation. The idea is to exploit the special structure of the Anonymous Zether relation (see [2, § 6.4]), in which *only two* of the ciphertexts (C_i, y_i) (under the public key D) are nonzero. For simplicity, we in fact restrict to the case of deposits and withdrawals, in which only one among the relevant ciphertexts is nonzero (i.e., per the definition of the relation).

Our idea is that the verifier, given the vector $(p_0, \ldots, p_{N-1}) \in \mathbb{F}_q^N$ computed in [2, Fig. 4], may use *both* the vectors $(p_0, \ldots, p_{N-1})$ and $(x^m - p_0, \ldots, x^m - p_{N-1}) \circ (v, \ldots, v^N)$—where $v \in \mathbb{F}_q$ is a *random* challenge chosen by the verifier—as exponent vectors in successive multi-exponentiations of the input vector $(C_i, y_i)_{i=0}^{N-1}$ of ciphertexts. By the structure of one-out-of-many proofs, the verifier's vector $(p_0, \ldots, p_{N-1})$ necessarily contains an m^{th} power of x at precisely one index, say $l \in \{0, \ldots, N-1\}$. It follows, therefore, that the m^{th} powers of x in the vector of scalars $(x^m - p_0, \ldots, x^m - p_{N-1}) \circ (v, \ldots, v^N)$ take the form $(v, \ldots, v^l, 0, v^{l+2}, \ldots, v^N)$, where here the l^{th} index—whichever it is (it's unknown to the verifier!)—is omitted or "exempted", and where the remaining indices are moreover nonzero and unpredictable to the prover. If the prover can cancel, from *this* multi-exponentiation, some lower-order terms in x, and nonetheless obtain a vector of plaintexts demanded by the verifier, it necessarily follows that the prover's plaintexts *other* than the l^{th} are in fact encryptions of zero. This is exactly what we need to prove. Note that our method requires just a *constant* number of multi-exponentiations, and so has complexity $O(N)$; in particular, it completely avoids use of the number-theoretic transform, which requires $O(N \log N)$ scalar multiplications. This particular procedure, and our improved verifier at large, can be seen at Firn's `WithdrawalVerifier.sol`.

Best-Ever Privacy

The final major technical contribution of Firn pertains to its sampling procedure for anonymity sets (i.e., its solution to the problem mentioned in [2, § 1.3]). A suitable selection procedure must simultaneously satisfy a handful of desiderata. For one, it must be highly efficient—say, logarithmic at worst—in the total state size of the network and in the total number of previous Firn users. Furthermore, it should return anonymity sets which are highly "representative", in the sense that their elements come from some distribution which is identical to—or, at least, as close as possible to—the distribution of *actual* transacting users. In particular, it should hold that these users' respective *times elapsed since last activity* closely match the distribution describing actual users' times since last activity. The key fact is that actual users are *much more likely* than (say) random users to have been *recently active*. Indeed, an actual withdrawer, with rather high likelihood, has deposited recently; a *random* depositor, with high likelihood, has *not* deposited recently. Selecting truly random depositors, in particular, would be a *bad* strategy, since the real depositor would "stand out" (in most instances) among the random depositors, as the sole account that was *recently* active. (In fact, an issue related to this affected a previous version of Monero [3], and was subsequently fixed.)

Our idea is thus to sample depositors *randomly*, but not *uniformly randomly*; rather, we sort depositors by time elapsed since most recent activity, and then sample them with a strong preference for *recently active accounts*. Specifically, we arrange depositors into a self-balancing binary search tree, sorted in ascending order by time of most recent activity. We then sample randomly from this tree, with an exponential preference for subtrees residing beneath nodes in the tree's right-most path. We achieve this in the following way, roughly. Writing L for the *height* of the tree, we first *uniformly* sample a *level* $l \in \{0, \ldots, L-1\}$. We then take l steps down the tree, walking *rightwards* each time, thereby reaching some account y, say. Having done this, we then sample *uniformly* from the subtree rooted at y, so obtaining some account, y_i, say. We repeat this sampling procedure until we obtain N distinct accounts $(y_0, \ldots, y_{N-1})$.

The anonymity sets which arise in this way are highly representative. For example, when $N = 16$, about 97% of them contain *the* single most recently-active account. In this light, even a user who *just deposited* may, upon withdrawing, nonetheless obtain strong plausible deniability, since most anonymity sets contain the most recently active user in any case. In practice, several technical matters arise. For example, we use *red-black trees*, which are, in general, not *perfectly* balanced; our sampling procedure must accommodate this. Separately, we bin users into "epochs" (see [1, § 2]); each node in the tree actually represents an epoch, and not an account. The code implementing our sampling procedure is available at `EpochTree.sol`.

References

[1] Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security (2020)

[2] Diamond, B.E.: Many-out-of-many proofs and applications to Anonymous Zether. In: IEEE Symposium on Security and Privacy. pp. 1800–1817 (2021)

[3] Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., Christin, N.: An empirical analysis of traceability in the Monero blockchain. In: Proc. Priv. Enhancing Technol. pp. 143–163 (2018)