Beam's Implementation of Mimblewimble: A Technical Rebuttal to Misinformation

Authors: The Beam Community
Date: August 31, 2025

A recent Substack article titled ***"Mimblewimble: A Critical Comparison of Privacy Approaches"*** presents numerous claims about Beam's implementation of Mimblewimble that contain significant technical inaccuracies and misleading characterizations.

As members of the Beam community with access to direct input from Beam developers (including Beam's protocol lead developer), we are sharing this detailed technical rebuttal to correct the record.

Fundamental Misunderstanding of Mimblewimble's Core Challenge

The above article opens with: *"True privacy doesn't arise from hiding data, but from its systematic destruction. What was never stored can never be compromised".* This statement, built upon specific characteristics of the Mimblewimble protocol, fundamentally misrepresents the most important privacy challenge in cryptocurrencies based on it.

Indeed, the author seems to lack the basic understanding of **Mimblewimble's main shortcoming: insufficient transaction graph obfuscation**. The most critical issue isn't what data gets stored long-term, but rather that transaction connections become visible to

the whole network the moment the transactions are broadcasted. Indeed, the protocol's ability to prune spent UTXOs provides no privacy benefits against an attacker actively listening to the network, because the transaction links were already revealed during the initial broadcast. This vulnerability becomes particularly acute during periods of low network saturation, when few transactions per block render Mimblewimble's built-in CoinJoin functionality less effective at obscuring transaction trails.

Beam developers invested significant effort to precisely address this core weakness of Mimblewimble. That's why some key architectural enhancements were developed to directly target this root vulnerability: first through the **strategic deployment of decoy outputs** (zero-value UTXOs) during Dandelion++'s stem phase, in order to complicate transaction graph analysis. And then, more fundamentally, through **periodic Lelantus shielded transactions** that completely sever linkability between inputs and outputs. These innovations address what truly matters -the prevention of transaction graph reconstruction- rather than only focusing on the protocol's long-term data retention characteristics, which -although interesting- are mistakenly elevated by the original article as the primary privacy concern.

References:

- Dummy UTXOs: https://medium.com/beam-mw/will-breaking-mimblewimbles-privacy-model-work-on-beam-9125bc2ee863
- Paper: https://github.com/BeamMW/beam/wiki/Transaction-graph-obfuscation
- Lelantus: https://beamprivacy.substack.com/p/fb6ed6cf-45a7-45c2-bc1c-fad9059017bd
- Paper: https://docs.beam.mw/Lelantus-MW.pdf
- About active attacks and transaction graph: https://medium.com/beam-mw/will-breaking-mimblewimbles-privacy-model-work-on-beam-9125bc2ee863

The Secure Bulletin Board System (SBBS): Clarifying the Record

The article claims: *"Every node in the Beam network stores a complete copy of this bulletin board. Thousands of nodes hold identical encrypted messages".*

This is misleading. While technically correct that all nodes store and relay all SBBS messages (that's precisely the decentralization and censorship-resistant strength of this messaging system), the article fails to mention the critical detail that **each message is only stored for 12 hours**, to prevent node and network bloat.

Beam's lead developer reminded us that this aspect was actually analyzed in detail when developing the SBBS feature. Indeed, despite the fact that all messages are encrypted and look like uniform random numbers, an attacker surveilling the whole network could indeed collect some metadata by time-matching the messages and the transactions. But the technical conclusions of the analysis of this kind of attack were that it is in fact a very minor risk. Indeed, the fact that a transaction is negotiated before it's sent is… obvious. The main

metadata leak could come from the attacker's ability to find the source of the SBBS message (the IP address of its sender). Yet, the same consideration exists in the transaction broadcast. And that's precisely why **Beam uses a modified Dandelion++ to broadcast transactions**.

All in all, the article creates inflated concern about metadata analysis, but fails to acknowledge that Beam actually implemented robust countermeasures to Mimblewimble's real main weakness.

UTXO Metadata: A Non-Issue Based on False Premises

The article claims Beam *"breaks [Mimblewimble's] principle by providing outputs with encrypted additional information"* and suggests that *"structurally, it's no longer the same"* and creates *"potential vulnerability."*.

This argument is fundamentally flawed. Beam's lead developer explains: *"Each UTXO looks like a uniform random number for the outside world. It can only be recognized and reverse-engineered by its creator. This principle holds for Beam, Grin and most other Mimblewimble cryptocurrencies. The only difference is that we at* ***Beam managed to use additional 64 bytes of information,*** *where the owner can store whatever it wants, without reducing the uniform random aspect of the UTXO."*

The article's concern about metadata creating future vulnerabilities is speculative at best. It acknowledges that "*the UTXO appears unchanged from the outside"*, but then goes on saying that there could exist some (vague and ill-defined) scenario where those particular 64 bytes of information could be identified and used as a unique UTXO identifier. Truth is that if somehow that were to be the case, then the whole UTXO could be reverse-engineered. And that would impact not only all Mimblewimble cryptocurrencies, but probably most of all cryptocurrencies!

Furthermore, the author ignores Beam's implementation of the different enhancements (such as the Lelantus transactions and the decoy outputs mentioned above) which actually significantly *strengthen* privacy protections beyond basic Mimblewimble.

Compliance Features: Mostly Theoretical and Misunderstood

The article devotes significant attention to Beam's *"compliance features*", claiming they *"relativize the principle of absolute privacy"* and that *"transparency becomes an imposed state"*.

This section is particularly misleading because those **"compliance features"** (in particular a feature allowing a read-only view of the wallet's balance and of its in/out-coming transactions) were actually… never developed! They exist solely as a theoretical option which was discussed at the beginning of the project.

Moreover the author's claim that *"anyone who activates auditability doesn't just expose their own side of a transaction, but automatically also that of the counterparty"* is incorrect in

practice. Indeed, if a user is obliged to report its transaction to the government (tax authorities, justice, etc.), they can do so by simply demonstrating their transaction history in the wallet, without the knowledge or consent of other involved users. And that is true for any cryptocurrency, as it has nothing to do with the existence or not of specific digital auditability features!

What actually exists today in Beam is an ***“owner key”*** which allows (among other things) a read-only view of a wallet’s balance (but not its transactions!). Such a viewkey on a wallet’s balance is useful for public wallets, donation funds, proof of reserves, etc. In addition, Beam also provides individual ***“payment proofs”***, which show and prove the content of *one* given transaction, and which can be shared one by one, transaction per transaction.

So the article's alarm about “compliance features” is largely based on a theoretical functionality that simply does not exist in the current implementation!

C++ Implementation: Security Expertise Over Language Hype

The article then claims C++ is *"notorious for buffer overflows, use-after-free bugs, and memory leaks"* and suggests Rust *"has become the de facto standard for security-critical blockchain projects"*. This reflects a superficial understanding of cryptographic engineering and security. Because security in privacy-focused systems isn’t determined by language popularity, but by the precision with which developers can control low-level execution.

Beam’s architecture and execution demands **absolute control over memory access patterns and timing side channels**, which are requirements critical both for maximum performance of complex cryptographic operations and for limiting risks such as timing attacks or cache-timing leaks. And these necessities perfectly align with C++’s granular hardware control.

This isn’t an ideological preference but an empirical necessity. And the article dismissing C++ for very general reasons seems to forget that all security-critical systems -from OpenSSL to most cryptographic libraries- **prioritize precise hardware control over syntactic safety**. As an example, that's the reason why many projects (and even other Mimblewimble projects the article praises) rely on the 'secp256k1' cryptographic library -which is written in C- for their core elliptic curve operations. If low-level control were truly "unpredictable," these industry-standard libraries wouldn’t exist.

Rust’s safety guarantees, while valuable for certain applications, also introduce architectural constraints that can conflict with cryptographic imperatives. By instance, its ownership model -designed to prevent memory errors- can force redundant data cloning or obscure low-level optimizations, directly undermining the constant-time execution and constant-memory access Beam requires. In fact, the problems Rust protects against are not sufficient for the code to be secure. **There's no silver bullet!** Even with all the "safety" features a language can provide, it is still possible to make mistakes. And in some situations, due to Rust’s higher-level nature, it can make it even harder -even for a skilled developer- to write a secure code.

At the end of the day, when implementing cutting-edge privacy cryptography, where a single timing leak can destroy privacy, **the developer's expertise and protocol-specific control outweigh language trends and popularity**. And Beam's experienced developers, with decades of work with multiple languages (including C++, Rust or Go), chose C++ with care and for good reasons.

Geopolitical Considerations: A Red Herring

The article's suggestion that Beam's Israeli origins raise concerns because Israel is *"a global center for surveillance technology"* is baseless speculation. A developer's nationality does not determine their political allegiance. Russian developers aren't Kremlin puppets by default, nor are American developers NSA agents!

In fact, the argument could even go the other way around: Regions with pervasive digital surveillance often cultivate **the most passionate and technically proficient privacy advocates**, as those who experience surveillance firsthand possess the strongest motivation to develop robust censorship-resistant technologies.

This line of argument has no technical merit and appears designed solely to create unfounded suspicion.

Ecosystem Fragmentation: A False Narrative

The assertion that Beam *"fragments the Mimblewimble community and thus weakens the entire ecosystem"* is nonsensical. True fragmentation occurs when projects *duplicate identical functionality*, not when distinct implementations address complementary use cases. Beam's architectural divergence from Grin and EPIC Cash doesn't fracture the ecosystem; it expands its utility frontier!

Consider Ethereum: Had we condemned its "fragmentation" of Bitcoin's ecosystem, we'd have stifled smart contracts, DeFi, and the entire Web3 revolution. **Diversity of implementation -where each project solves specific privacy challenges- is the engine of progress**, not a weakness.

Conclusion: Different Approaches, Same Goal

The original article presents Beam's design choices as compromises on privacy, forcing a certain narrative into its analysis and distorting or misrepresenting certain aspects to make Beam look bad or suspicious. And this is particularly striking when in reality those specific design choices represent thoughtful enhancements which precisely address the main Mimblewimble weaknesses.

Beam's approach isn't about "compromising" privacy principles but rather about strengthening them in practical implementations. The project has made deliberate choices to address real-world privacy challenges that the other minimalist implementations leave unaddressed.

The article's criticism largely stems from misunderstandings (or voluntary misrepresentation?) of Beam's actual implementation, confusion between theoretical and released features, and an overly rigid interpretation of what constitutes "true" Mimblewimble.

For those genuinely interested in privacy technology, it's worth recognizing that different projects address different aspects of the privacy challenge. **Beam's contributions to transaction graph obfuscation and practical privacy for real-world applications represent unique and valuable advancements in the field** - advancements that this article fails to acknowledge while propagating numerous technical inaccuracies.

Rather than viewing these different approaches as competing philosophies, the privacy community would benefit from recognizing them as complementary efforts toward the shared goal of creating genuinely private digital transactions.

To learn more about Beam, visit beam.mw

Download the Desktop Wallet from beam.mw/downloads

See the source code on github.com/BeamMW

Or join the community on socials: X, Telegram, Discord, etc.