Nostrust: Attestable GDPR-Compliant Relays for Decentralized Publishing

Komron Aripov
Brown University

Prithvi Oak
Brown University

Abstract

Nostr [9] is a decentralized protocol for web publishing, but its use of third-party relays for data storage and processing poses privacy and GDPR compliance challenges. This project addresses these concerns by implementing an attestable Nostr relay and compatible Nostr client using Intel Software Guard Extensions (SGX) [7] and encrypted databases. Intel's SGX provides a Trusted Execution Environment (TEE) that enables secure and remotely verifiable data processing on untrusted servers, while encrypted databases ensure that user data remains confidential during storage. This approach guarantees transparency and compliance with GDPR regulations, giving users greater control over their data within a decentralized system.[1]

1 Introduction

Online publishing and microblogging platforms, such as Twitter, have been instrumental in the popularization of public online discourse. With that popularity, in recent years, we have seen the rise of alternative decentralized services that mimic Twitter's functionality, such as Mastodon and BlueSky, through the use of decentralized protocols such as ActivityPub [14] and AT Protocol [8] respectively. Nostr is one such service, consisting of a lightweight JSON-based protocol described in Nostr's Implementation Possibilities (NIPs), which external servers, called relays, implement to specification.

The design of Nostr permits clients to choose any of multiple such relays for storage and dissemination of their data for various reasons, including but not limited to: censorship resistance, avoidance of data monopolization, and enhanced user control. Ultimately, however, much of the power rests within the relay server owners and administrators, who are able to institute arbitrary rules for the data collected and disseminated through their relays. Therefore, the full advantages of Nostr as a decentralized service are only available to those who self-host their relays and are able to disconnect from untrustworthy, but potentially popular, third-party relays.

In the worst case, third-party relays that are opaque to end-users will violate important regulations, such as GDPR's Right to Deletion. Without guarantees on data handling or deletion, Nostr's protocol introduces risks that erode user trust and can lead to violations of GDPR's requirements for privacy, transparency and user consent.

[1]All authors contributed equally.

Our prototype relay, Nostrust, implements the Nostr protocol within SGX – a secure hardware primitive provided by Intel. SGX lets a process initialize a secure address space region called an enclave. Any code executed within the region is then protected from the host operating system, including memory reads and code modification. Therefore, Nostrust aims to protect user data from a potentially malicious host while adhering to the NIP specifications and additional GDPR guarantees. In principle, using SGX, we can guarantee any number of additional desirable properties, such as content moderation or additional regulation adherence, such as HIPAA.

However, Nostrust must rely on host OS provided primitives, such as filesystem and networking, to communicate with its clients. Furthermore, clients must be sure that the relay they are communicating with is running on genuine SGX hardware. To achieve these goals, Nostrust uses multiple standard and SGX-provided cryptographic primitives. Namely, we use SGX-provided attestation to verify that the relay is running unmodified code, SGX-provided sealing to encrypt user data at rest, as well as signature verification over Nostr events as required by the NIP specifications.

In order to fully prevent the host OS from selectively servicing certain user requests and not others, it is recommended to also implement TLS over TCP. Nostrust also modifies the Nostr protocol to use HTTP requests instead of WebSockets, which should otherwise be used to maximize the number of users who are able to interact with the relay. Both of these properties are very desirable but out of scope for this prototype relay to reduce implementation complexity on niche hardware. Lastly, Nostrust implements only several key NIPs, described below, for similar reasons.

2 Background

The Nostrust prototype has three key components. Firstly, it implements the Nostr protocols through the NIP specification. Secondly, it uses and implements wrappers around key SGX CPU instructions that enable attestation and sealing.

```
{
  "id": <hex sha-256 of event>,
  "pubkey": <hex of key of event creator>,
  "created_at": <unix timestamp in seconds>,
  "kind": <integer between 0 and 65535>,
  "tags": [
    [<arbitrary string>...], // ...
  ],
  "content": <arbitrary string>,
  "sig": <hex of the signature of event>
}
```

Figure 1: Nostr Event object schema

Lastly, it guarantees data deletion as accorded to by GDPR. We explore each of these components, as well as how they interact, in detail.

2.1 NIPs

NIPs are the building blocks of the Nostr protocol, akin to a Request For Comments (RFCs) or Python Enhancement Proposals (PEPs). However, unlike RFCs or PEPs, NIPs are not binding and follow a decentralized adoption approach, which leads to different relays offering varying degrees of support for a given NIP.

The most important and defining NIP is NIP-01 [10], which specifies the key Event object. The Event object (1) contains arbitrary user-supplied content, alongside the user public key and a signature over the data for verification. An important feature of the Event object is the `kind` field, which identifies to the relay what kind of event it is handling and is the basis of many other NIPs. For example, `kind = 1` is for plaintext public messages and is specified in NIP-01, while `kind = 5` is for event deletion requests and is specified in NIP-09.

2.2 Intel SGX

Intel SGX, first introduced in 2015 with the introduction of Intel's 6th generation Skylake microarchitecture, has been pivotal in helping software developers handle sensitive consumer data [5]. A number of projects, such as Ryoan [6] and Haven [1], handling user data in the cloud on untrusted servers, have since started using SGX as a trusted hardware primitive. SGX provides a number of guarantees for Nostrust that are summarized below.

Memory protection. SGX keeps the contents of the memory within an enclave private. Any data provided by the application resides encrypted within memory, and probing requests outside the SGX result in a fault. While the data is within cache, it is protected by the CPU with access controls. This protection is at a physical level – SGX uses Enclave Page Cache (EPC), which maintains a private virtual memory mapping for SGX.

Attestation. SGX includes a CPU instruction to derive two key identifiers: a measurement and a quote. A measurement is the hash of the initial state of the program running within the enclave, including its stack, heap, allocated pages and source code. A Report is a structure containing the measurement and a signature over it signed by a key known only to the SGX and with a chain of trust leading to Intel's Root Certificate Authority (CA). This signature is known as the quote. With the combination of measurement and quote, an SGX relay can attest to running on genuine SGX hardware and executing genuine Nostrust code.

Sealing. To stop data loss across restarts, user data must eventually be stored on disk. If stored as-is, however, this exposes the data to inspection and tampering by a potentially malicious host OS. This is where SGX Sealing is helpful. SGX deterministically derives a unique sealing key based on the static Root Sealing Key known only to itself and an additional parameter. In the case of Nostrust, we can specify this additional parameter to be the enclave's measurement, tying it directly to the code running on the enclave. Any data stored on disk can be encrypted with this key, and is only decrypted if the subsequent enclave has an identical measurement, i.e. untampered code.

It is important to note that a malicious host OS can perform a rollback attack, whereby it doesn't return the latest encrypted blob. This can be circumvented through use of a trusted "data freshness" service, but is outside the scope of this project.

2.3 GDPR

The General Data Protection Regulation (GDPR) is a European law that ensures protection of personal information and outlines the principles that any entity processing user data must follow [3].

- **Article 17** of the GDPR details the *Right to erasure*, which allows users of a service to delete their personal data, either due to withdrawal of consent, unlawful processing of data, lack of necessity of the data for the purposes for which it was collected, or other relevant circumstances.
- **Article 5** specifies that personal data must be processed "lawfully, fairly, and in a transparent manner" with re-

spect to the user. This means that a user's data can only be used for specified and legitimate purposes.

For users of a online publishing platform, the GDPR enforces protections on the way the service handles their data; the user must be allowed to delete all their personal data from a service, and the service cannot be using their data for any purpose other than what is specified to the user.

For a user on a Nostr relay, these fundamental parts of the GDPR are not guaranteed to apply. The user has no way of ensuring the legitimate use or successful deletion of their data. NIP-09, which specifies event deletion requests, is also an optional component of the protocol, which means there is no strict obligation for a relay to specify its practices of GDPR deletion.

3 Design

Nostrust introduces a novel *trusted relay*, which ensures lawful and transparent processing and deletion of user data.

3.1 Nostrust relay

The relay runs in SGX to allow for trusted execution and attestation. As depicted in 2, SGX runs within an untrusted host OS, ensuring that the OS cannot access sensitive data or computations within SGX enclaves. As a result, the Nostrust relay requires **untrusted runners** to communicate to the relay and the database. To ensure security from untrusted entities—the client, runners, file runner, the OS—the relay uses techniques to protect and validate the data.

- **Message verification** through signatures on messages ensures that information that the client sends the relay through untrusted runners is correct and unmodified. This provides the guarantee that the messages a relay receives from a user are, indeed, from that user.
- **Cryptographic sealing** ensures that the data received by the file runner is unreadable; the data is only decrypted within the Nostrust relay, not outside of it. In order to seal and unseal the data, a sealing key is obtained from the enclave. Cryptographic shredding ensures that data stored in the file runner can never be decrypted, even if copied by the untrusted file runner.

Similar to the original design of Nip-01, the Nostrust relay processes messages from the client to create new events, add subscriptions, remove subscriptions, and send requested events. With the implementation of NIP-09, secure GDPR deletion can be done, and a modified NIP-11 can provide both relay information and an attestation measurement to the Nostrust user.

The Nostrust relay design ensures generalizability on multiple properties; with the SGX enclave providing guarantees on code attestation and data sealing, frameworks can be build on the relay to enforce and guarantee policies beyond GDPR deletion. For example, an identical design can be used to ensure content moderation to users of the relay.

Two separate components were built to handle communication with and storage for the relay—a prototype Nostrust client and a file runner for data storage.

3.2 Nostrust client

A simple client was built to manage and test communication with the Nostrust relay. A user of the client can input 6 kinds of requests to interact with the relay:

- **post** to post new content on the relay
- **follow** to subscribe to other users
- **unfollow** to unsubscribe to other users
- **get** to retreive the user's feed based on their subscriptions
- **delete** to delete all posted content (GDPR deletion)
- **info** to retrieve information and an attestation measurement from the relay.

Each user is assigned a public and private key for identification and signing of their messages. Inputted requests are converted into *client messages* that are sent to the relay, and information is received from the relay in the form of *relay messages* that are transformed into human-readable format.

3.3 File runner

A custom file runner enables blob storage services for the relay. Since SGX restricts filesystem access for the relay, this file runner allows for the storage and access of sealed relay data. The data stored by the file runner includes (1) all the existing and undeleted events received by the Nostrust relay, and (2) all the active user subscriptions.

The relay interacts with the file runner either to pull and unseal data from the runner or to seal and send the updated blob to the runner. For Nostrust, data is encrypted via the the AES-GCM-128 primitive derived from the 16-byte sealing key provided by SGX. In the prototype stage, the encryption and decryption steps are triggered through a secret API path call.

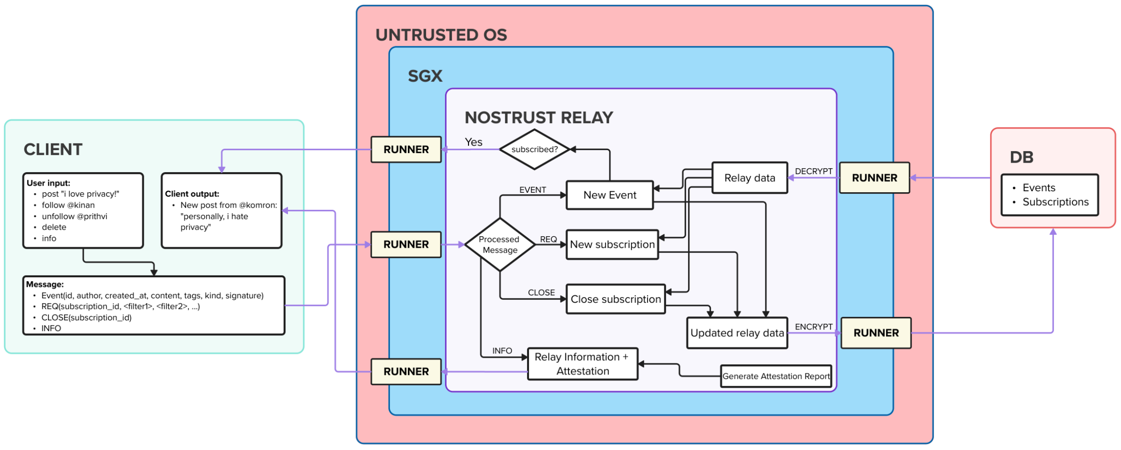

Figure 2: Flow diagram for Nostrust Client-Relay-DB interactions

4 Implementation

The Nostrust relay, client, and filerunner were all built in Rust. Combined, approximately 1500 lines of code were written across each component of the project.

Rust is chosen for this project due to rich library of cryptographic, web, and serialization frameworks. For example, `serde` [12] is used throughout the project to comply with NIP-01's JSON schemas. Additionally, Rust has a number of actively maintained SGX libraries, of which we chose to use Fortanix's Enclave Development Platform (EDP) [4]. The EDP provided an SGX compilation target for Rust, as well as a number of utility libraries that wrapped around SGX CPU instructions. Fortanix also provides runners to communicate with a number of untrusted interfaces such as TCP sockets and standard output.

Crucially, this project required specialized hardware. To successfully run Nostrust, it is necessary to have an SGX-enabled machine, drivers and correct BIOS settings. When such a machine is unavailable, SGX Simulator Mode [2] is a viable alternative for everything other than full attestation and sealing verification, as a simulator cannot produce keys with Intel's chain of authority. Full setup information can be found in the artifact `README` file.

5 Evaluation

We run a number of qualitative and quantitative tests that ensure that baseline requirements for Nostrust as a trustworthy, GDPR-compliant relay are met. Namely, we ensure full implementation of three baseline NIPs running on SGX hardware. Furthermore, we confirm that the attestation and sealing mechanisms fail for modified binaries.

5.1 Components

Baseline. We successfully implement Nostr's main NIPs: NIP-01, NIP-09 and NIP-11. The main objects, such as the Event object described in NIP-01, are exposed to both the relay and the client as a core library, including additional functionality for verification of Event signatures according to the Schnorr standard for the curve `secp256k1` [15]. These are added as unit tests to the library.

The Nostrust prototype, however, modifies the communication protocol from a standard relay to use HTTP requests rather than WebSockets. SGX, as a unique compilation target, does not compile for all major web framework libraries with WebSocket implementations and requires separate implementations of networking primitives that would've been outside the scope of this project. An HTTP implementation eases developer effort for serialization and deserialization of Nostr objects.

As a qualitative test, we observe the relay servicing multiple clients, including all of the relevant commands. Messages are verified and the relay compiles and runs within the SGX target on SGX hardware, while clients run outside the SGX target.

Sealing. While Fortanix EDP provides runners to interface with major untrusted interfaces such as network sockets and standard output, it does not do so for the filesystem. We successfully implement an untrusted filerunner that enables filesystem functionality via a separate server for saving and loading relay state.

We observe that decryption panics via a sealing key derived on a binary different from the one that encrypted it – we choose to symbolically modify the event deletion mechanism. This test requires an operational filerunner, client, relay and a separate interface for sending requests to the secret API path, such as cURL.

Attestation. We use SGX CPU reporting functionality to obtain the enclave's measurement, which is exposed to the user via an information request defined in NIP-11. We observe that any change in the relay binary generates different measurements, allowing users to verify that the Nostr is running unmodified code.

Further steps to fully verify the attestation require large additional infrastructure changes. Our tests were conducted on a machine with the SGX1 instruction set, which only supports Enhanced Privacy ID (EPID)-based remote attestation. EPID is an API service provided by Intel and consumed by clients that wish to verify the authenticity of SGX reports. EPID has been phased out starting September 29, 2024. Current remote attestation mechanisms involve SGX Data Center Attestation Primitives (DCAP), that are only supported on the SGX2 instruction set. DCAP provides similar guarantees to EPID, but requires either a self-hosted attestation server or a trusted third-party for full verification.

5.2 Limitations

While the Nostrust prototype demonstrates the potential for a secure, GDPR-compliant Nostr relay using Intel SGX, there are aspects of its current design and implementation that warrant consideration.

A potential attack from a malicious OS that is not handled by the current Nostrust implementation is the selective servicing of user requests or other communications via runners. This is something that implementing TLS for communication could prevent but is out of the scope of the project due to implementation complexity. A similar limitation is faced in using WebSockets as per the Nostr Protocol. The primary reason for the implementation complexity is the target-specific nature of crates like tokio [13], which would otherwise enable smooth deployment of these features. Future work on Nostrust relies on building supporting SGX architecture for these components.

Due to the lack of filesystem access under SGX, a separate untrusted file runner was used to communicate with data storage, sealing the data to guarantee safety from untrusted services. However, there is a high efficiency cost of sealing and unsealing a whole database that would become more pertinent if Nostrust were scaled for more users and clients. Looking ahead, more efficient ways to retrieve and store information on databases might be considered. Storage solutions employing cryptographic techniques such as CryptDB [11] might allow for a more scalable Nostrust architecture, while providing security on user data.

Lastly, it is important to note that SGX is not a panacea for all the risks of trusting a Nostr relay. While SGX provides strong guarantees for memory protection, attestation, and data sealing, it cannot fully protect against all attack vectors. For instance, vulnerabilities in SGX itself, risks associated with the underlying hardware, or insufficient infrastructure for verifying the integrity of the relay can still compromise the system.

References

[1] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding applications from an untrusted cloud with haven. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 267–283. USENIX Association, 2014.

[2] Intel Corporation. Usage of simulation mode in sgx-enhanced applications, December 2024. Intel Developer Training Article.

[3] Council of the European Union. General data protection regulation. https://gdpr-info.eu/, 2018.

[4] Fortanix. Enclave development platform (edp). https://edp.fortanix.com/, 2024.

[5] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Carlos Rozas, Vinay Phegade, and Juan del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *HASP 2013: Hardware and Architectural Support for Security and Privacy*. Intel Corporation, 2013.

[6] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 533–549. USENIX Association, 2016.

[7] Intel Corporation. Intel® software guard extensions (intel® sgx) overview. https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html, 2024.

[8] Martin Kleppmann, Paul Frazee, Jake Gold, Jay Graber, Daniel Holmgren, Devin Ivy, Jeromy Johnson, Bryan Newbold, and Jaz Volpert. Bluesky and the at

protocol: Usable decentralized social media. In *Proceedings of the ACM Conext-2024 Workshop on the Decentralization of the Internet (DIN '24)*, pages 1–12, Los Angeles, CA, USA, December 2024. ACM.

[9] Nostr Project. Nostr: A better internet is possible. https://nostr.com/, 2024. Accessed: 2024-12-13.

[10] Nostr Protocol. Nostr implementation possibilities (nips). https://github.com/nostr-protocol/nips, 2024. Accessed: 2024-12-13.

[11] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: A practical encrypted relational dbms. Technical Report, February 2012. Technical report detailing the design and implementation of CryptDB, a practical encrypted database system.

[12] Serde Project. Serde: A framework for serializing and deserializing rust data structures. https://serde.rs/, 2024. Accessed: 2024-12-13.

[13] Tokio Project. Tokio: An async runtime for the rust programming language. https://tokio.rs/, 2024. Accessed: 2024-12-13.

[14] World Wide Web Consortium (W3C). Activitypub: A decentralized social networking protocol. https://www.w3.org/TR/activitypub/, 2018. W3C Recommendation, January 23, 2018.

[15] Pieter Wuille and contributors. Bip-340: Schnorr signatures for secp256k1. https://bips.xyz/340, October 2020. Bitcoin Improvement Proposal (BIP) describing the Schnorr signature scheme for secp256k1.