Orchhit: User-Oblivious Social Networking

JIM MCCANN[*], TCHOW llc
BRIAN SAGHY, my email was down for three hours

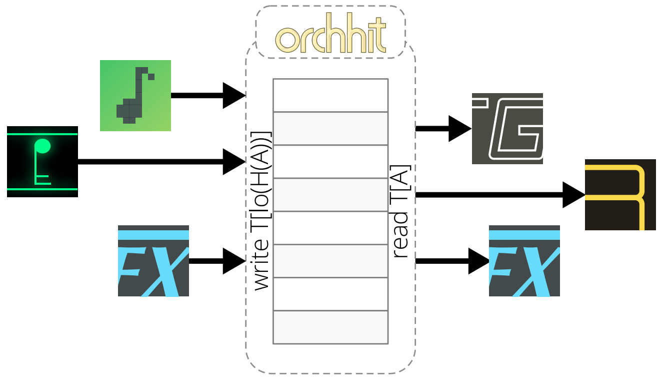

Fig. 1. Our user-oblivious social network, Orchhit, allows sharing of orchestral-hit-sound status updates in a user-oblivious way, thanks to a preimage-writable table construction.

In this paper, we describe the unique infrastructure used by our new user-oblivious social network, Orchhit (Figure 1). This infrastructure uses a constant-sized storage to support basic status sharing for an unlimited number of users; allows instant client-side account creation and deletion; and is immune to server-side snooping. Since we are magnanimous co-founders, we reveal our infrastructural secrets in this tell-all publication.

CCS Concepts: • **Security and privacy** → *Privacy protections*; Hash functions and message authentication codes; • **Networks** → *Social*.

Additional Key Words and Phrases: privacy, failed social networks, overly casual writing, isn't it unfortunate that cryptozoology has nothing to do with cryptosystems?

1 INTRODUCTION

Modern social network web *page* (site) companies face three major problems: acquiring new users, providing server resources to support existing users, and repressing their own inescapable desire to sell users' private information. We present a novel social network architecture – user-oblivious social networking – that uses cryptographic primitives to mitigate all three of these problems.

Particularly, our user-oblivious social network backend provides a status sharing platform and the following guarantees:

(1) Instant client-side account creation and deletion without server contact (thus, no way for operators to determine the number of accounts).
(2) Friends lists are never stored on the server in a way that can be read by operators (thus, no way for operators to determine the social network of accounts).
(3) Uses a constant amount of server storage.
(4) Provides no way to distinguish status updates from random (or user account) data, unless status updates are improperly designed.

[*] ix@tchow.com

2 BACKGROUND

Learning from the mistakes of others would only slow us down.

3 METHOD

Our user-oblivious social network construction is built on a cryptographic hash function $H(X) : \mathbb{Z}_2^* \to \mathbb{Z}_2^{2B}$ which maps arbitrary-length bit-strings to fixed-length bit strings of length $2B$ in a way that is hard to invert and does not induce any correlations between output bits (along with some other important properties that we can't be bothered to look up right now).

3.1 Server

The social network server is responsible for persistent storage of a 2^B-entry table of B-bit storage locations, T. This table is initialized with random bits.

The server provides two interfaces, `read` and `write`, to the client to manipulate the table. Read allows a client to retrieve the table value at a given B-bit address:

$$\begin{aligned} &\texttt{read}(A : \mathbb{Z}_2^B) : \\ &\quad \textbf{return } T[A] \end{aligned} \tag{1}$$

Write allows a client to set the table value at any address for which it knows a B-bit hash preimage:

$$\begin{aligned} &\texttt{write}(P : \mathbb{Z}_2^B, V : \mathbb{Z}_2^B) : \\ &\quad T[\texttt{lo}(H(P))] \leftarrow V \end{aligned} \tag{2}$$

Where $\texttt{lo}(B) : \mathbb{Z}_2^{2B} \to \mathbb{Z}_2^B$ is the function that returns the low B bits of a $2B$-bit value.

These two primitives – `read` and `write` – are the only things that the server implements in Orchhit – the remainder of the social network relies on client operations exclusively.

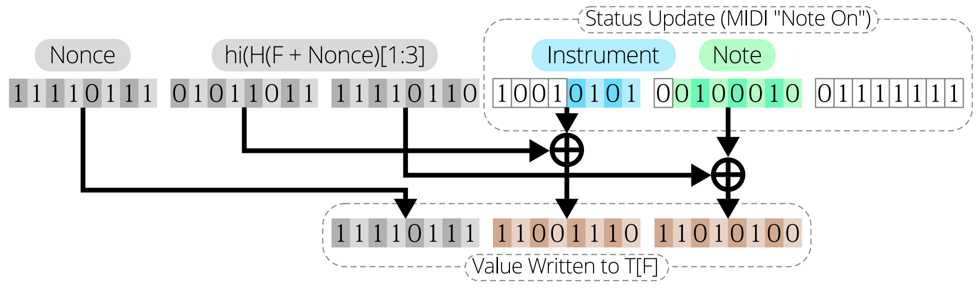

Fig. 2. The status format in Orchhit is a MIDI note-on message whitened by a nonce.

3.2 Client

Notice that the server operations described above allow secure publish-subscribe interaction between clients – one client can pick a value P to write status updates to and publish value $\texttt{lo}(H(P))$ to enable other clients to read these updates. This interaction forms the basis of our platform's social networking operations.

Login / account creation. To start using our social network, a client picks a passphrase, P; computes a secret key, $K \equiv H(P)$, by hashing the passphrase; and derives a follow key, $F \equiv \texttt{lo}(H(\texttt{lo}(K)))$, by hashing the low bits of the secret key.

Notice that this process does not require any server communication or storage. Note, also, that the high bits of the secret key will never be sent to the server.

Account deletion. Account deletion can be performed by forgetting the passphrase.

Status updates. To publish a status update, the client writes the update using its secret key:

$$\begin{aligned} &\texttt{publish}(U): \\ &\quad \texttt{write}(\texttt{lo}(K), U) \end{aligned} \tag{3}$$

Friends list. To store or retrieve the ith element of its friends list, the client uses addresses, P_i, derived from its secret key, obscuring the contents with a value, X_i, derived from the extra-secret upper bits of its secret key:

$$\begin{aligned} X_i &\equiv \texttt{lo}(H(\texttt{hi}(K) + i)) \\ P_i &\equiv \texttt{lo}(H(\texttt{lo}(K) + i)) \end{aligned}$$

$$\begin{aligned} &\texttt{getFriendAddress}(i): \\ &\quad \textbf{return } \texttt{read}(\texttt{lo}(H(P_i))) \oplus X_i \end{aligned} \tag{4}$$

$$\begin{aligned} &\texttt{setFriendAddress}(i, A): \\ &\quad \texttt{write}(P_i, A \oplus X_i) \end{aligned} \tag{5}$$

Notice that no provision is made for variable-length friends lists. In our implementation, every user has exactly four friends.

4 PROPERTIES

The above construction is simple but provides some very useful properties which make it hard to deduce anything about the social network by inspecting a snapshot of its storage.

User Account Obliviousness. Since all of a user's account data (their friends list) is xored with a passphrase-dependent stream, it is impossible to distinguish user accounts from empty (initialized-to-random) storage.

Constant Storage. Rather than consuming ever-increasing amounts of storage, the server's table remains fixed-size for the life of the network. As the number of users grows, the user experience will gracefully degrade as friends list entries and statuses get over-written by hash collisions. As the designers of the internet understand, this sort of "gentle failure" is much preferable to a hard failure, and may even lead to self-regulation, as users will quit using the (apparently flaky) system.

Status Update Obliviousness. As long as status updates are IID[1], and fill the entire encoding space, status updates stored in the table are also indistinguishable from random bits.

At present, this is an implementation detail that the client needs to manage[2].

5 IMPLEMENTATION

Our user-oblivious social network, Orchhit, is live at http://orchhit.com. It provides users the ability to push orchestral hit sounds to their followers and to follow up to four friends. Status messages are, therefore, MIDI note-on messages, which are whitened using a nonce and a hash value in a way that is somewhat flawed, Figure 2, though red-teaming this is left as an exercise to the reader .

For our implementation, we chose $B = 24$ as a reasonable compromise between usability and security[3]. As a hash function our implementation uses `SHA-1` (truncated to 48 bits) because implementations are readily available and because we clearly lack cryptographic acumen.

As a bandwidth saving measure, the `read` call implemented by our server provides the option to defer return until the value is different from a provided value. This technique, termed *long polling*,

[1]"Locally owned and responsibly sourced."
[2]We *certainly* didn't get this wrong in our client.
[3]Is 2^{24} a sufficiently large number? Of course it is! It's higher than most people can count even if they use binary and all their fingers and toes.

avoids some bandwidth costs and *probably* doesn't open the system to any weird timing attacks.

6 FUTURE WORK

The astute among you may have realized by now that selecting a secure value for B may be impossible, especially as compute efficiency seems to be growing much faster than storage efficiency. One solution might be to use a pearl-diver construction to provide proof-of-work along with read requests[4].

For some social network designs, allowing only four friends may seem limiting. However, this limit can easily be overcome by realizing that your definition of "friend" is not sufficiently narrow (or by keeping multiple accounts open).

Unfortunately, our system is vulnerable to several side-channel information disclosure attacks. Anyone able to observe the network traffic to and from the server – e.g. the social network operator themselves – can estimate the type of individual storage locations by observing the read and write behavior over time. This information may be used to determine the number of users and – potentially – the contents of their status updates. In our present network, this hazard is largely mitigated by the fact that status updates are just orchestral hit sounds so, like, chill out folks.

7 CONCLUSION

In this paper, we described a construction for a user-oblivious social network based on a preimage-writable table[5].

Readers are encouraged to try our social network at http://orchhit.com. Philosophically speaking, you either already have, or can never truly have, an account.

We hope that our work points the way forward for a bold new set of web services that use cryptographic primitives to make scaling easy and monetization nearly impossible.

[4]This may enable blockchain-backed automated fulfillment services to support big-data-centric AI-first omnichannel retailtainment; enhanced, of course, by a sustainable and authentic brand story. Yes, just put the venture capital money over there.

[5]Which we have actively avoided looking up prior work on, preferring to treasure the illusion of novelty.