Zero-Friction Identity Bootstrapping via SSH Key Reuse

Ramprasad Anandam Gaddam
Vouch Protocol Project
https://vouch-protocol.com
ram@vouch-protocol.com

January 2026

Abstract

This paper presents a method for bootstrapping cryptographic identity from existing SSH keys registered with platforms such as GitHub. Rather than requiring users to generate new keypairs and register new identities, the system leverages SSH keys already associated with developer accounts. This "piggyback" approach enables immediate adoption of the Vouch Protocol with zero additional key management burden. We formalize the hybrid verification model, specify the key resolution protocol using platform APIs, and analyze security properties and trust assumptions.

This paper is part of the Vouch Protocol Defensive Disclosure Series. Full index: https://vouch-protocol.com/docs/disclosures/

Keywords: SSH keys, identity bootstrapping, zero-friction onboarding, GitHub API, hybrid verification

1 Introduction

Adopting new cryptographic identity systems typically requires generating new keypairs, establishing key distribution channels, and building trust in new identifiers. This adoption friction limits uptake of otherwise valuable security systems.

We observe that over 100 million developers already possess cryptographic keys: their SSH keys registered with GitHub, GitLab, and similar platforms. These keys are:

- Already generated and stored on developer machines
- Already registered with trusted platforms
- Already associated with established identities
- Already used for high-value operations (code deployment)

We present a hybrid verification system that "piggybacks" on existing SSH keys, enabling zero-friction adoption of the Vouch Protocol.

1.1 Contributions

- Hybrid verification model combining SSH keys with Vouch DIDs
- Key resolution protocol using platform public key APIs
- Security analysis of trust assumptions
- Fallback mechanism for non-platform users

2 Background

2.1 SSH Key Authentication

Definition 1 (SSH Key Pair)**.** *An SSH key pair* (sk, pk) *where:*

- sk*: Private key stored in* `~/.ssh/id_ed25519`
- pk*: Public key registered with remote services*

SSH authentication proves possession of sk through challenge-response signature [Ylonen and Lonvick, 2006].

2.2 GitHub Key API

GitHub exposes registered SSH keys via public API:

```
GET https://api.github.com/users/{username}/keys

Response: [
  {"id": 12345, "key": "ssh-ed25519 AAAA..."}
]
```

This API requires no authentication and returns all SSH public keys for any user.

3 Hybrid Verification Model

3.1 Verification Priority Chain

When verifying a Vouch token claiming identity I:

1. If I is a DID: Resolve DID document, verify using DID key
2. If I is a GitHub username:
   (a) Fetch keys from `api.github.com/users/{I}/keys`
   (b) Attempt signature verification against each returned key
   (c) If any key validates: Verified as GitHub user I
3. If I is a GitLab username: Similar process with GitLab API
4. Otherwise: Verification fails

3.2 Token Format for SSH-Based Identity

```
{
  "iss": "github:alice",
  "iat": 1704844800,
  "exp": 1704845100,
  "vouch": {
    "version": "1.0",
    "payload": {"action": "approve_pr", "pr": 42}
  }
}
```

The `iss` claim uses prefix `github:` or `gitlab:` to indicate platform-based identity.

3.3 Verification Algorithm

1. Parse issuer: $(\mathsf{platform}, \mathsf{username}) = \mathsf{parse}(\mathsf{iss})$
2. Fetch keys: $K = \mathsf{API.getKeys}(\mathsf{platform}, \mathsf{username})$
3. For each $pk \in K$:
   (a) If $\mathsf{Verify}(pk, \mathsf{payload}, \sigma) = 1$: Return verified
4. Return failed

4 Security Analysis

4.1 Trust Assumptions

Property 1 (Platform Trust)**.** *The system trusts the platform API to return authentic public keys for the claimed username.*

This trust assumption is reasonable because:

- GitHub/GitLab are widely trusted for code hosting
- The same keys are used for production code deployment
- Key registration requires account authentication

Property 2 (Key Authenticity)**.** *If GitHub reports key pk for user U, then U controls the corresponding sk.*

4.2 Threat Analysis

Threat: Platform compromise. If GitHub is compromised, false keys could be returned. Mitigation: Secondary verification sources, key pinning.

Threat: Account takeover. If a GitHub account is compromised, the attacker can add keys. Mitigation: Key age verification, multi-source verification.

Threat: API unavailability. If GitHub API is down, verification fails. Mitigation: Key caching with TTL.

4.3 Comparison with Native DIDs

Property	SSH Piggyback	Native DID
Setup friction	None	Key generation required
Trust anchor	Platform	Self-sovereign
Platform dependency	Yes	No
Key rotation	Platform-managed	Self-managed

Users can start with SSH piggyback and migrate to native DIDs later.

5 Implementation Considerations

5.1 Key Caching

To reduce API calls and handle unavailability:

$$\mathsf{cache}[\mathsf{username}] = (K, t_{\mathsf{fetched}}, \mathsf{TTL}) \tag{1}$$

Recommended TTL: 1 hour for active verification, 24 hours for fallback.

5.2 Rate Limiting

GitHub API rate limits: 60 requests/hour unauthenticated, 5000/hour authenticated. Caching is essential for high-volume verification.

6 Related Work

Keybase linked cryptographic keys to social identities but required explicit registration.

Web Key Directory (WKD) provides domain-based key discovery for email; our approach uses platform APIs.

GitHub Sigstore Integration uses OIDC for keyless signing; our approach reuses existing SSH keys.

7 Implementation

Reference implementation available at https://github.com/vouch-protocol/vouch.

Setup command: `vouch git init` configures repository to use SSH key signing with automatic platform resolution.

8 Conclusion

This paper presented SSH-based identity bootstrapping for zero-friction adoption of the Vouch Protocol. By leveraging existing SSH keys and platform APIs, users can begin signing attestations immediately without generating new keys or registering new identities. The hybrid verification model provides a practical on-ramp to cryptographic identity for the developer community.

Prior Art Declaration

This document is published as a defensive prior art disclosure under the Creative Commons CC0 1.0 Universal Public Domain Dedication. The methods and systems described herein are hereby released into the public domain to prevent patent monopolization.
Any party implementing similar functionality after the publication date of this document cannot claim novelty for patent purposes.
Reference Implementation: https://github.com/vouch-protocol/vouch

Cryptographically Signed Document

Signed by: Ramprasad Anandam Gaddam (`github:rampyg`)
Verify: https://v.vouch-protocol.com/p/pad008

This document's authenticity can be verified by computing its SHA-256 hash and checking against the signature registered at the verification URL above.

References

T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, IETF, January 2006. URL https://www.rfc-editor.org/rfc/rfc4253.