Sharing is caring; Centralized PKI and masking for multi-organization blockchain sharing

Lef Ioannidis

January 4, 2020

1 Introduction

Private blockchains are an oxymoron, neither a fully decentralized blockchain nor a centralized database of transactions. They are only viable as long as they keep the best of both worlds; the scalability and automation found in a blockchain, as well as a central point of cross-cutting, high-level control across the entire system. This document introduces *Authoritarian*, a centralized Public Key Infrastructure (PKI) and policy enforcing system, that allows multiple organizations to share one distributed ledger.

*Authoritarian*acts both as a gateway to blockchain participation and a bootstrapping system for new nodes. As the goals of *Authoritarian*are relatively broad, some assumptions were made in order to limit the search space of possible design solutions. First, a universe of *organizations* share access to the same blockchain. Those organizations can have logical subgroups which might overlap and vaguely resemble Unix user groups. The lowest order principal in *Authoritarian*is a single node, which represents a single machine in the organization. Nodes can keep secrets from each other, which is why all transaction metadata are stored anonymized and encrypted. At the same time, exchanges can be performed across organizations and verified by all nodes, based on a set of policies specific to the organization. Every transaction block will be checked by a committee to verify the policies have been withheld, the committee elected using a proof-of-stake protocol. In addition, a policy constraint solver can be fairly useful in detecting logical errors in the declarative specification using automatic reasoning, similar to AWS IAM [1].

2 Design

*Authoritarian*is composed of two centralized services; a name server(*DNSSec*) and an authentication server(*Auth*), as well as a distributed ledger. *DNSSec* and *Auth* handle bootstrapping, naming, authentication, policy control and privacy. Both are globally accessible to all nodes, either by VPN or the open internet. A VPN has the advantage of a reduced attack surface, as well as some useful non-repudiation properties for detecting and blacklisting bad nodes. Conversely, an open-internet approach is open to both untargeted and targeted attacks and susceptible to a spectrum of DoS attacks as well. However, a VPN without unsafe entrypoints is an operational assumption regarding the deployment of the system and rather optimistic. *Authori-*

*tarian*has end-to-end encryption build-in everywhere after bootstrapping and makes no assumptions regarding the underlying network topology. The only assumptions we make are the eventual availability of the *DNSSec* and *Auth* servers, as well as the reachability of at least 51% of the nodes as is usually the case for distributed ledgers.

2.1 Principals

Access control is defined in terms of principals. A three-tier hierarchy of organizations (`ON`), groups (`OU`) and nodes (`CN`) allows for fine-grained access control. Figure 1 shows the principal hierarchy for Acme Inc, with two groups and a total of five node machines. Every organization has a dedicated administrator (`ON admin`) who implements the bootstrapping process.

A web-of-trust of X509 CAs guarantees end-to-end encryption and non-repudiation across principals. Bootstrapping gives Organizations an intermediate CA and a dedicated DNS zone in the *DNSSec* server and after that, individual nodes can authenticate with the organization and automatically register and participate in *Authoritarian*. New groups and nodes can issue certificates for themselves by proving ownership of a DNSSec domain by completing a challenge, similar to Let's Encrypt certbot [2]. A secret will be served on the node machine via HTTPS, if this secret can be reached by the certbot querying the DNS domain of the node, a client-certificate will generated and signed by the ON intermediate CA.

2.2 Bootstrapping

A unique self-signed root CA is used to sign two level-1 intermediate certificates for the *Auth* and

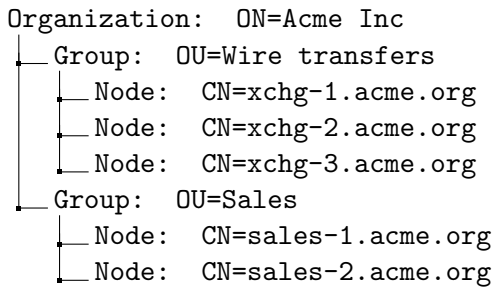

Figure 1: Principals tree; an organization with two groups and five total nodes, each node is a *DNSSec* domain

DNSSec servers, then stored offline in a safe, air-gaped location. It will only be used again to rotate the keys in *Auth* and *DNSSec* in an annual or biannual basis. The ON admin has to manually generate a Certificate signing request (CSR) and have it signed by the *Auth* intermediate CA through a side-channel, in order to receive a level-2 intermediate CA for the `ON`.

Each organization is a logical namespace, represented by a unique DNS zone, `*.acme.org` in 1 for example. To bootstrap an organization, a static DNS zone needs to be manually added once, to the global *DNSSec* record. Assuming that is done, nodes in this organization can register under that zone themselves by using a DynDNS update command [3] authenticated by a pluggable machine authentication protocol specified by the organization administrator.

An organization incorporates their internal machine authentication policy to ensure no outside machines can register under their Zone. This is usually done via a firewall. A rather open firewall policy is to allow every machine in the corporate IP subnet to send a DynDNS update to the global *DNSSec* server. A better one is to seed every machine with a UUID and whitelist those UUIDs in the firewall, before al-

lowing DynDNS domain registration. In either case, every organization identifies their machines in a different way which is opaque to *Authoritarian*but familiar to the `ON admin`. It is crucial to realize every machine that can register itself to the *DNSSec* server will also be able to receive a client-certificate from the *Auth* server and participate in distributed consensus committees.

2.3 Enforcing policies

Policies in *Authoritarian*are specified in a declarative language similar to AWS IAM policies [1]. Figure 2 shows a policy for hiding the transaction history of one organization from their competitors, while fig. 3 shows a policy for allowing software engineers to get paid.

The *Auth* server stores all policies. For each block, a new random committee is elected. Each member of the committee will receive all relative policies from the *Auth* server and verify that they have been enforced for each transaction in a block, before committing it to the blockchain. There are no meta-permissions in place, in other words all policies in the system are publicly readable. Since it might not be ideal to send a policy to the principal from which it protects from, the following heuristic is applied; *A policy will not be sent to a principal which it directly references*. For example, the policy in 2 will not be sent to nodes in EvilCorp and thus will not leak the existence of this rule.

```
{
  "Name": "Hide history from the
     competition",
  "Statement": {
    "Effect": "Deny",
    "Principal": {
      "ON" : "EvilCorp",
      "OU" : "*",
      "CN" : "*"
    }
    "Action": "read",
    "Resource": "log::*"
  }
}
```

Figure 2: Deny policy; Acme Inc. hides their transaction history from EvilCorp indiscriminately

```
{
  "Name": "Pay software engineers",
  "Statement": {
    "Effect": "Allow",
    "Principal": {
      "ON" : "Acme",
      "OU" : "SWE",
      "CN" : "*"
    }
    "Action": "accept",
    "Resource": {
      "ON" : "Acme",
      "OU" : "HR",
      "CN" : "*"
    }
  }
}
```

Figure 3: Accept policy; Acme HR is allowed to pay software engineers (SWE), notice a resource can be either a regex or a principal.

3 Distributed Ledger

After authenticating, nodes can access the transaction log, perform transactions and be elected in committees and vote for a new block. A committee is randomly selected and implements the

proof-of-stake algorithm described below. Every member of the random committee checks all transactions in the block to verify they match the relevant policies received from the *Auth* service. It might be tempting here to use the *Auth* here to pick committees in a methodical, centralized way. However, this has the risk of single point failure in the *Auth* resulting in bad blocks being committed or the DoS of the whole blockchain. The DoS danger remains present, as the inability of nodes to receive policies from *Auth* will not allow new blocks to be committed, but it would be preferable if a compromise in the *Auth* or *DNSSec* would not allow bogus blocks to be accepted.

3.1 Proof-of-Stake

A simple proof-of-stake (PoS) algorithm can be used to add blocks to the blockchain by means of a committee, elected randomly across its members. similar to [4], each node will perform a federated lottery to see if he receives a winning ticket. Every node with a winning ticket will become a committee member and receive the candidate block as well as all the relevant policies from the *Auth* server. If the transactions in the block match the policies, and all committee members agree that the block is a reflection of the distributed ledger, the block will be then committed to the blockchain. The common "nothing-at-stake" issue [5] can be mitigated in the PKI implementation by ostracizing bad actors. A suggested protocol is to penalize nodes which accept all forks indiscriminately by temporary certificate revocation.

3.2 Crypto

Nodes will communicate via an encrypted, non-repudiated channel with other nodes and the *Auth* after bootstrapping X509 client certificates. Certificates are tied to the domain name that was requested by the node via *DNSSec*. The X509 certificates wrap around an Elliptic Curve25519 [6] public key, generated and used for encryption and signing with the BoringSSL library, made by Google. BoringSSL is not only used in about a billion Android and chrome devices, but the particular Curve25519 implementation in it has been formally verified [7]. Unlike standard SSL/TLS where a variety of algorithms can be offered from the server, here only Curve25519 will be used for PKI, as this reduced the cryptographic attack surface considerably and there's no backwards compatibility requirement for the proposed protocol.

3.3 Masking transactions

One of the requirements for the private blockchain is to enable private transactions to be stored and verified. When verifying a candidate block, a policy similar to 2 dictates all relevant metadata should be hidden from adversaries. This will be done by salting and hashing the principals, as well as encrypting the transaction metadata. The salt will be a global counter maintained by *Auth* but incremented by every node using a Lamport clock[8]. Amounts, timestamps and other metadata will be encrypted using symmetric `aes-256-gcm` and the secret principals as the key. The invariant here is that if the sender and recipient of a transaction are known, the amount and time of the transaction can be decrypted and used to verify the transaction in a block. *Auth* server will serve as an unmasking

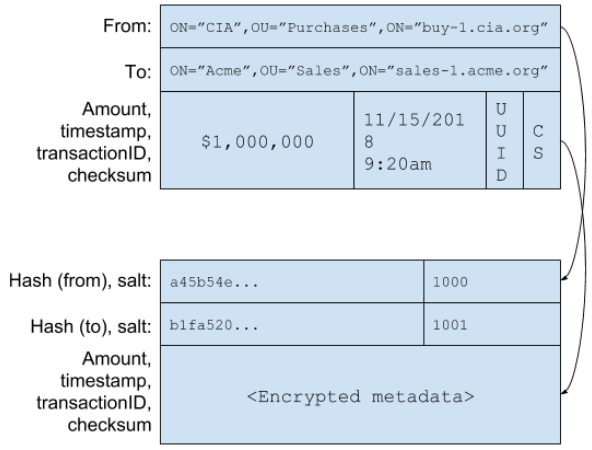

Figure 4: Transactions in a block are masked and encrypted. The From/To fields are hashed with a random salt which is included and the metadata are encrypted with `aes-256-gcm` and $key = hash(From, To)$ so only when the sender and recipient are unmasked the amount is visible.

mechanism and will keep a mapping of all masks to principals for this purpose.

Figure 4 shows how transactions are masked and stored inside a block. All transaction will be masked by default, and only "read" queries as in 3 will be unmasked by the *Auth* service. Masking everything by default adds a little complexity to how policies are handled though, since policies reference unmasked principals. Again the *Auth* server can be helpful by transforming policies to their masked counterparts.

3.4 Masking policies

*Authoritarian*will need to enforce policies even without knowing the principals to which they apply, an ability necessary for block verification. *Auth* will mask policies to correspond to the block a committee handles. Figure 5 shows how 2 will be masked to apply to the masked transaction 4. Notice how the masked sender and recipient in 5 are masked with the same salt as in 4.

```
{
  "Statement": {
    "Effect": "Deny",
    // Acme Corp (masked)
    "PrincipalMasked" : ["a45b54e
        ...", "1000"]
    "Action": "read",
    // Evil Corp (masked)
    "ResourceMasked"  : ["bf45432
        ...", "1002"]
  }
}
```

Figure 5: Masked version of policy 2

4 Conclusion

*Authoritarian*tries to walk the line between decentralized consensus and federated access policies. It requires minimal bootstrapping once for every organization, and afterwards individual nodes can register and participate themselves. The privacy of the participants is preserved by using the *Auth* server as an anonymizing proxy. In addition, high-level policies are enforced by the nodes which verify every transaction in a block before committing to the blockchain, in a scalable and sustainable way. Some disadvantages are that the *Auth* server is a bottleneck for masking policies when a new committee is elected and a DoS attack on *Auth* can lead to no new blocks being committed. This problem can be solved with increased availability of *Auth* and DDoS mitigation by means of proof-of-work

in the client requests to *Auth*. It is important to verify all policies in the nodes in a distributed way though, otherwise a compromised *Auth* can lead to bogus blocks being committed to the blockchain. All in all, *Authoritarian*is an opinionated approach to a broad problem and relies on cryptography and distributed systems algorithms to address it. It does not suffer from a variety of spoofing, tampering, information disclosure and repudiation attacks and is end-to-end encrypted with minimal bootstrapping for new nodes. Additional extensions in the future should include formally verifying parts of *Auth* as well as parts of the distributed consensus such as the Lamport clock salt selection, as well as tools to help `ON admins` incorporate their machine authentication to *Authoritarian*.

References

[1] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. Luckow, N. Rungta, O. Tkachuk, and C. Varming, "Semantic-based automated reasoning for aws access policies using smt,"

[2] J. Aas, "Let's encrypt: Delivering ssl/tls everywhere," *Let's Encrypt*, vol. 18, 2014.

[3] P. V. et al, "Dynamic updates in the domain name system (dns update)," RFC 2136, RFC Editor, April 1997.

[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 51–68, ACM, 2017.

[5] V. Buterin, "Slasher: A punitive proof-of-stake algorithm," *Ethereum Blog URL: https://blog. ethereum. org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm*, 2014.

[6] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*, pp. 207–228, Springer, 2006.

[7] A. E. J. Philipoom and J. G. R. S. A. Chlipala, "Simple high-level code for cryptographic arithmetic–with proofs, without compromises,"

[8] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.