Threshold Cryptosystem Based Decentralized Secure Protocol

Andrew Bednoff, Rust Software Engineer,
Darkblock.io

August 29, 2022

Decentralized secure pairing-based threshold cryptosystem (re-encryption protocol) for collaborative/collective decryption (one-key-of-many decryption scheme) and signatures.

Main use case: Decentralized Digital Rights Management (DRM) for content (unlockable digital assets) access control.
Also can be used for developing distributed and decentralized message (and data) exchange platforms and communication systems.

https://darkblock.atlassian.net/wiki/spaces/TECH/pages/72941569/Decentralized+secure+pairing-based+threshold+cryptosystem+re-encryption+protocol+for+collaborative+collective+decryption+one-key-of-many+decryption+scheme+and+signatures

Explicit requirements:

1. Decentralized and distributed, i.e. every piece of system should be able to accept separated common responsibility (distributed key generation, storing a key fragments, content files, metadata, participate in encryption and decryption processes) and contribute to collective shared work.

2. Communication transport channels should be secured via symmetric encryption for both sides

3. Symmetric secret keys for securing communications should be derived via key exchange session (ECDH) and not stored locally

Explicit requirements:

4. Usage of hybrid cryptosystems: stored locally asymmetric service and carrier nodes private keys and private key shares/fragments stored on carrier-nodes can be hardened, encrypted via symmetric encryption and symmetric keys can be stored in secure persistent local key storage (SGX sealed keys, TPM). Stored locally symmetric keys should be secured via asymmetric cryptosystems.

5. For the performance reasons content files should be secured by symmetric cipher

6. Content files will be encrypted by symmetric cipher and its secret key should be secured by asymmetric one-key-of-many scheme, i.e. encrypted via threshold cryptosystem via main public key, secret keys for future decryption should be splitted/separated into fragments via pairing based cryptosystem (BLS signature scheme), transferred (see point #2, #3) and stored (see point #4) securely

7. Plaintext symmetric key or decrypted content shouldn't be leaked through memory in processes of ciphertext (with symmetric key) decryption meeting or content decryption and re-encryption phase, thus stream symmetric ciphers (like ChaCha 2.0) preferred, and advanced techniques of memory protection (mlock on app compilation phase, Intel SGX usage) for more improved security are advisable.

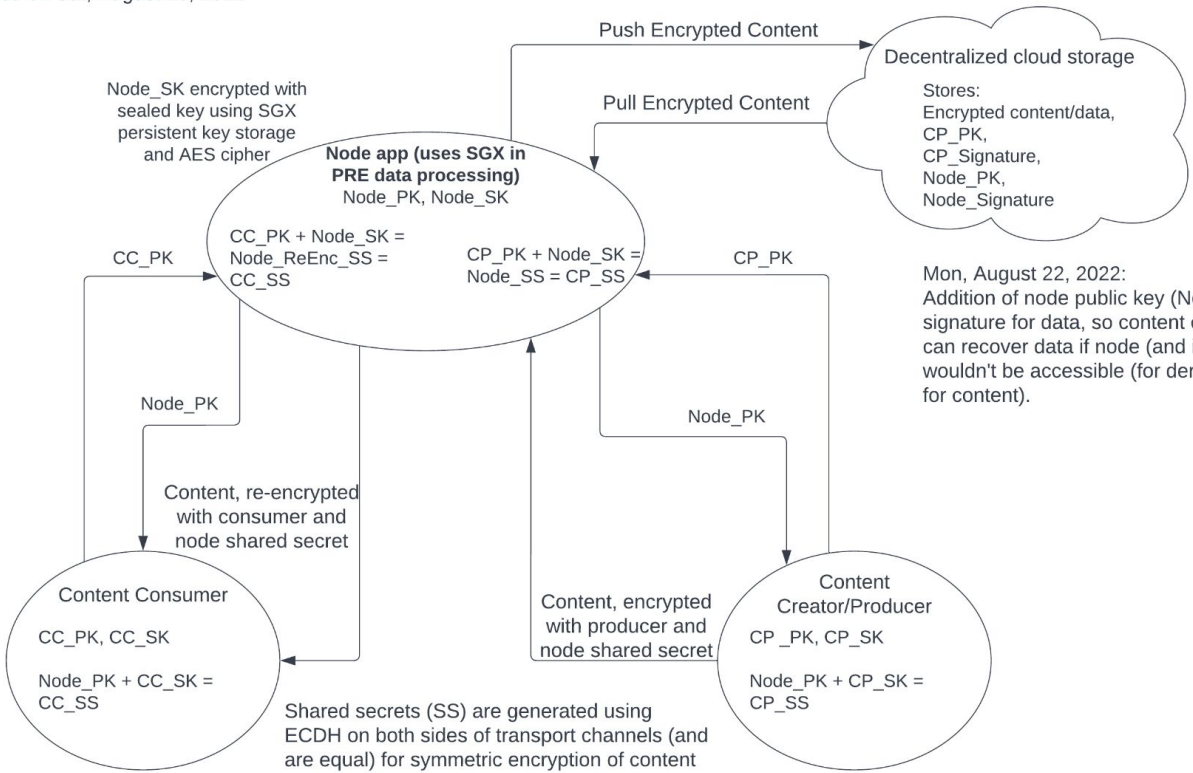
Implicit requirements:

1. The cryptosystem should be compliant with definition, characteristics and requirements of zero-knowledge proof protocol. All point listed above in explicit requirements are compliant with zero-knowledge proof protocol requirements.
2. Content creator should be able to recover/restore encrypted data, i.e. retrieve data from distributed file storage, decrypt it, re-encrypt it, and post to distributed file storage again. (But only if content creator still has an ownership of content and ownership transfer of digital asset not yet or wasn't performed.)

First iteration: ***Simply Secure SGX based re-encryption protocol***

https://darkblock.atlassian.net/wiki/spaces/TECH/pages/71958529/Simply+Secure+SGX+based+re-encryption+protocol

Created on Sat, August 20, 2022



Mon, August 22, 2022:
Addition of node public key (Node_PK) and node signature for data, so content creator (producer) can recover data if node (and its public key) wouldn't be accessible (for deriving shared secret for content).

Shared secrets (SS) are generated using ECDH on both sides of transport channels (and are equal) for symmetric encryption of content using AES block cipher or other cipher (as ChaCha 2.0 stream cipher, suitable for data streaming using QUIC).
I.e. ECDH used only for deriving the symmetric keys for content encryption/decryption/re-encryption and for securing the transport for public keys exchange without sharing symmetric keys itself.

First iteration: ***Simply Secure SGX based re-encryption protocol***

Issue: the content is strictly sticked to the initial node, which receives and posted content to distributed file storage, due to the derived symmetric key during key exchange (ECDH) session, thus key derivation for content encryption is dependent from public/private key pairs of both sides, and thus decryption node should be the same.

Solution:
We should secure symmetric key by asymmetric scheme, but secret key sharing cannot be avoided in any case.
Then let's take a look on secret sharing schemes and distributed key generation. Secret key can be splitted/separated into key fragments. This will increase the complexity of secret key retrievement and makes it impossible. Minimal threshold should be set for secure multi-party computation for decryption. This will prevent decryption of content outside of trusted content provider.

Next step:

The solution is:
Usage of pairing based cryptography and group signatures to make one-key-of-many scheme and threshold signatures cryptosystem.
BLS (Boneh–Lynn–Shacham) signatures scheme is the most suitable one.

Papers:

Short signatures from the Weil pairing

Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme

Simple and Efficient Threshold Cryptosystem from the Gap Diffie-Hellman Group

Next step:

Already existed libraries, written in Rust:

https://crates.io/crates/pairing

https://crates.io/crates/bls12_381

https://crates.io/crates/blsttc

https://crates.io/crates/bbs

https://crates.io/crates/bzte

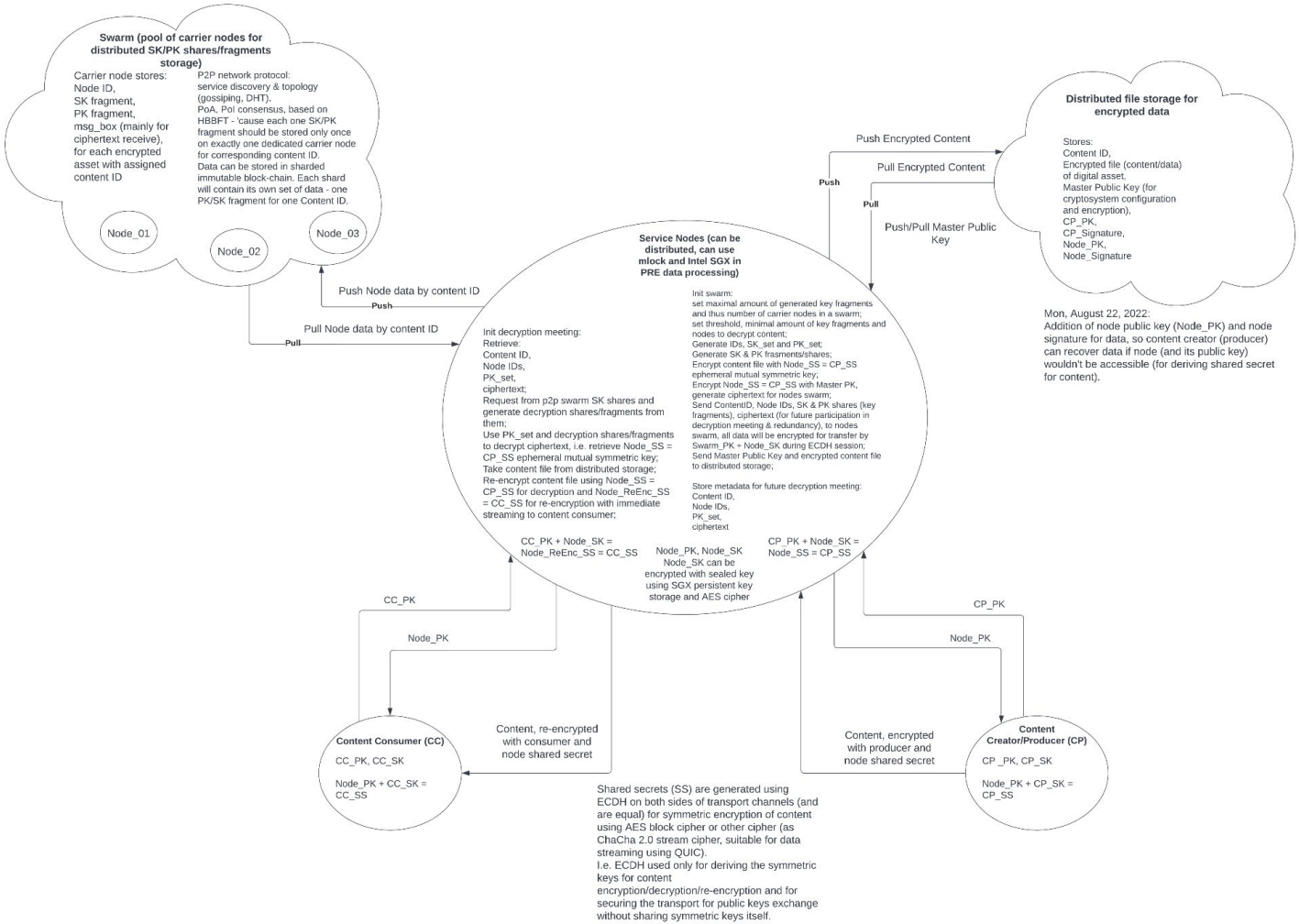
https://crates.io/crates/nube

https://crates.io/crates/bls_dkg

Second iteration: ***Threshold Cryptosystem Based Decentralized Secure Protocol***

https://darkblock.atlassian.net/wiki/spaces/TECH/pages/72941569/Decentralized+secure+pairing-based+threshold+cryptosystem+re-encryption+protocol+for+collaborative+collective+decryption+one-key-of-many+decryption+scheme+and+signatures

Created on Thu, August 25, 2022



Simplified description of protocol and crypto scheme:

Init swarm:

- set maximal amount of generated key fragments and thus number of carrier nodes in a swarm;
- set threshold, minimal amount of key fragments and nodes to decrypt content;
- Generate IDs, SK_set and PK_set;
- Generate SK & PK fragments/shares;
- Encrypt content file with Node_SS = CP_SS, ephemeral mutual symmetric key;
- Encrypt Node_SS = CP_SS with Master PK, generate ciphertext for nodes swarm;
- Send ContentID, Node IDs, SK & PK shares (key fragments), ciphertext (for future participation in decryption meeting & redundancy), to nodes swarm, all data will be encrypted for transfer by carrier node PK + servicing node SK during ECDH session;
- Send Master Public Key and encrypted content file to distributed storage;

Store metadata for future decryption meeting:

- Content ID,
- Node IDs,
- PK_set,
- ciphertext

Simplified description of protocol and crypto scheme:

Init decryption meeting:
Retrieve:

- Content ID,
- Node IDs,
- PK_set,
- Ciphertext;

- Request from p2p swarm SK shares and generate decryption shares/fragments from them;
- Use PK_set and decryption shares/fragments to decrypt ciphertext, i.e. retrieve Node_SS = CP_SS ephemeral mutual symmetric key;
- Take content file from distributed storage;
- Re-encrypt content file using Node_SS = CP_SS for decryption and Node_ReEnc_SS = CC_SS for re-encryption with immediate streaming to content consumer (and content decryption on consumer end);

Simplified description of protocol and crypto scheme:

Carrier node stores:

- Node ID,
- SK fragment,
- PK fragment,
- msg_box (mainly for ciphertext receive),

for each encrypted asset with assigned content ID

Distributed file storage, for encrypted data, stores:

- Content ID,
- Encrypted file (content/data) of digital asset,
- Master Public Key (for cryptosystem configuration and encryption),
- CP_PK (to verify the content origin, integrity and content recovery possibility),
- CP_Signature (to verify the content origin, integrity),
- Node_PK (to verify content origin from authorized source and content recovery possibility),
- Node_Signature (to verify content origin from authorized source)

All data exchange, i.e. communication channels (transports), between servicing nodes and fragments carries nodes, servicing nodes and distributed file storage, servicing nodes and Content Producers, servicing nodes and Content Consumers, will be encrypted for transfer by (servicing node, carrier node, Content Producer, Content Consumer, correspondingly) PK + SK pairs and derived ephemeral symmetric key, during ECDH key exchange session.

Proposed threshold cryptosystem demo and POC

- Demo run of POC for proposed threshold cryptosystem application
- Measuring performance via running benchmarks for block and stream ciphers, written in Rust, inside the Intel SGX enclave (using AWS Nitro Enclave and SGX Rust compiler backend target for standalone application).

Standalone app using Intel SGX on local CPU vs. AWS Nitro Enclaves:

I've found a solution how to eliminate vendor locking, i.e. dependency on AWS cloud. The Rust compiler has a backend compiler target for SGX from Fortanix. It can be used for application compilation to make standalone executables which can utilize Intel SGX technology on a local CPUs. So, it can be used to develop the Darkblock node or to compile the dedicated components/subsystems/service of the node, which are used for working with sensitive data in memory (unencrypted data during decryption and re-encryption and unencrypted secret or private keys).
Mlock compilation option also can be used on a compile-time phase to protect memory of application during run-time phase.

https://edp.fortanix.com/docs/

Proposed threshold cryptosystem characteristics:

- The proposed threshold cryptosystem doesn't reveal in any memory (RAM or ROM) unencrypted data during decryption and re-encryption phase. Usage of symmetric stream ciphers prevents memory leaks and memory dumps by third parties.
- All secrets remain private and stored only locally on network nodes.
- If stored locally, secrets and private keys should be encrypted.
- Symmetric keys, for encrypted transport communication channels between nodes, and symmetric key, for content encryption by content producer, decryption and re-encryption for content consumer, are derived during public key exchange session (ECDH) and not even stored locally.
- Private keys for decryption of content secret and content itself are stored as a fragments only in a swarm of fragment carrier nodes.
- Mlock and Intel SGX can be used for even more improved security of application memory during the run-time phase.

August month goals:

- Formalize the protocol, cryptosystem and hence the future development process steps.
- Provide the POC for proposed threshold cryptosystem.
- Provide the demo of benchmarks for block and stream ciphers implementations in Rust.
- Perform the demo run of POC for proposed threshold cryptosystem application and cipher benchmarks inside the Intel SGX enclave (using AWS Nitro Enclave and SGX Rust compiler backend target for standalone application).

“If there's a vision - there's an action.”

So, formalized vision of what exactly we should do, gradually, is really important to moving forward.

And now we have a vision of the future path, the protocol and how to make a standalone app, and hence how to make a Darkblock node with decentralized networking capabilities.

Q&A:

Any questions?

Thank you!