



ExamCrypt: A Secure Examination Paper Management System Using Hybrid Cryptography and Blockchain Auditing

Ashish Shedge ^{a*}, Harshal Patond ^a, Praneet Revandkar ^a

^a Department of Computer Engineering, Smt. Indira Gandhi College Of Engineering, Mumbai, Maharashtra, India

ABSTRACT

A question paper that leaks early, or is quietly altered on its way to the examination hall, invalidates the whole assessment, yet many institutions still move papers through e-mail, removable drives and shared folders that protect the file only at the first hand-off. This paper presents ExamCrypt, a web-based system that manages a question paper from upload to timed release under cryptographic control. Each paper is hashed with SHA-256 and encrypted with a per-document AES-256-GCM key, and that key is wrapped with RSA-OAEP. The digest and every later event are anchored in an Ethereum-compatible ledger, so the database administrator is no longer the only witness to what happened. Release needs approval from a separate role and falls inside a scheduled window, and each released copy is watermarked with its recipient. We describe the architecture, role model, data schema and algorithms, and report micro-benchmarks of the cryptographic path on documents of 0.1 to 25 MB. On the test machine a 10 MB paper was encrypted in about 15.2 ms and decrypted in about 15.2 ms, and all 3,000 simulated tampering attempts were detected. Ledger latency and end-to-end tests belong to the ongoing work and are not claimed here.

Keywords: Examination security; hybrid encryption; AES-256-GCM; RSA-OAEP; SHA-256; blockchain audit trail; role-based access control; time-locked release

1. Introduction

Question papers are among the few assets an examining body cannot afford to lose control of. A paper that leaks a day early, or one that is altered between the paper setter and the examination hall, damages the assessment no matter how carefully everything else is run. In many institutions the handling chain is still largely manual: the setter e-mails a file or hands over a USB drive, a moderator reviews a copy, and the examination section prints another. Password-protected archives and shared folders with role-based permissions improve matters, but they protect the file at one point only and leave little evidence of who handled it afterwards.

Three weaknesses recur in such arrangements. Confidentiality usually depends on a single shared password or on the goodwill of an administrator. Nothing ties the released paper to the one that was approved, so a modification can pass unnoticed. And the logs, when they exist, sit on the same server as the documents and can be edited by the people they are supposed to watch. Release timing is typically enforced by convention rather than by the system.

This paper presents ExamCrypt, a secure examination paper management system that treats these four properties (confidentiality, integrity, accountability and controlled release) as built-in behaviour rather than optional settings. Our contributions are as follows.

A five-stage paper lifecycle (upload, hash, encrypt, log, approve) in which every transition is enforced by the application and recorded.

A hybrid protection scheme in which a per-document AES-256-GCM key is wrapped with RSA-OAEP, while the SHA-256 digest of the plaintext is anchored on a ledger so that integrity can be checked independently of the database.

Approval-gated, time-bounded release with a role check on every request, and a per-recipient watermark on each released copy.

A relational schema that links every recorded action to a ledger reference.

Measurements of the cryptographic path on documents of 0.1 to 25 MB and a tamper-detection experiment.

The rest of the paper is organised as follows. Section 2 reviews related work, Section 3 states the threat model, Section 4 describes the design, Section 5 gives the algorithms, Sections 6 and 7 cover the implementation environment and the preliminary evaluation, and Sections 8 and 9 discuss limitations and conclude.

2. Related work

2.1 Hybrid encryption and integrity checking

Encrypting bulk data with a fast symmetric cipher and protecting only the key with a public-key scheme is the standard way to combine the speed of one with the key-distribution convenience of the other. The public-key side traces back to the RSA system (Rivest, Shamir, & Adleman, 1978), the symmetric side to the Advanced Encryption Standard (NIST, 2001). Leela and Vinod (2018) describe a hybrid cryptosystem built around digital signatures, which is the closest general reference for the document-protection pattern we follow. Two details matter in practice. Galois/Counter Mode (Dworkin, 2007) provides an authentication tag together with encryption, so a modified ciphertext is rejected at decryption. For the key itself, OAEP padding (Bellare & Rogaway, 1994) is preferred over textbook RSA. Separately, the Secure Hash Standard (NIST, 2015) defines SHA-256, which gives a fixed-length digest suitable for detecting any change to a document.

2.2 Ledger-based audit trails

A blockchain is an append-only sequence of blocks linked by hashes (Nakamoto, 2008); altering an old entry changes every digest after it. Ethereum added programmable contracts (Wood, 2014), and permissioned platforms such as Hyperledger Fabric add identified participants and policy control, which suits institutional settings (Androulaki et al., 2018). Two earlier systems are worth noting because they keep the bulky data off the chain. Zyskind, Nathan and Pentland (2015) use a ledger as an access-control manager over data stored elsewhere, and Ramachandran and Kantarcioglu (2017) use smart contracts to record data provenance. ExamCrypt follows the same off-chain/on-chain split: papers stay encrypted in a document store, and only digests and event records go to the ledger.

2.3 Access control and timed release

Role-based access control grants permissions to roles and assigns roles to users (Sandhu, Coyne, Feinstein, & Youman, 1996). The zero-trust model in NIST Special Publication 800-207 (Rose, Borchert, Mitchell, & Connelly, 2020) adds that no request should be trusted because of where it comes from; each one is authenticated and authorised on its own merits. For timed release, Rivest, Shamir and Wagner (1996) proposed time-lock puzzles, where unlocking requires a fixed amount of sequential computation. That approach ties the opening moment to the solver's speed rather than to a wall-clock time, so it does not fit an examination that must open at a stated minute. We therefore use a policy-based time gate backed by key custody, and we are explicit about what this does and does not guarantee in Section 8.

2.4 Gap addressed by this work

In the literature we reviewed, hybrid encryption, ledger auditing, role-based governance and timed release appear as separate ideas. We did not find a design that combines them, together with approval workflow and watermarking, for the specific problem of examination paper handling. Table 1 summarises how common practices compare on these properties, based on the arrangements described in Section 1.

Table 1 - Properties of typical paper-handling practices compared with ExamCrypt.

Practice	Encrypted at rest	Integrity check	Tamper-evident log	Approval gate	Timed release
E-mail or USB transfer	No	No	No	No	No
Password-protected file	Partial	No	No	No	No
Role-based file server	Partial	No	Partial	Partial	No
ExamCrypt (proposed)	Yes	Yes	Yes	Yes	Yes

2.5 Threat model and design goals

We consider four kinds of adversary. An external attacker may read stored files or traffic. An insider who holds a legitimate role (an examiner, approver or invigilator) may try to act outside it, for example to read a paper before release. A privileged operator, such as a database or server administrator, may try to replace a ciphertext, edit an approval or rewrite a log entry. Finally, a recipient of a released paper may pass it on. We assume that TLS protects data in transit, that the key-management component on the server has not been compromised, and that the ledger is operated by a permissioned set of institutional nodes. Coercion, side channels, compromised exam-time endpoints and analogue leaks such as photographing a printout are outside the scope of this work.

From this model we derive five goals: (G1) confidentiality of papers at rest and in transit; (G2) integrity, meaning any change to a paper is detectable; (G3) auditability, meaning every action leaves a record that the actor cannot later rewrite; (G4) governed access, with no implicit trust for any principal; and (G5) controlled release, meaning no access before approval and the scheduled time. Table 2 maps each adversary to the controls that address it.

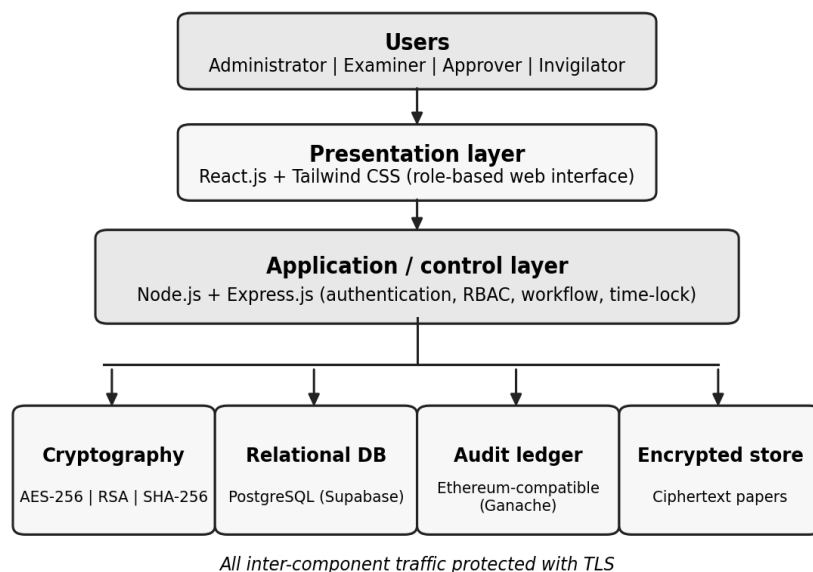
Table 2 - Adversaries and the controls that address them.

Adversary	Main controls	Residual risk
External attacker	AES-256-GCM at rest; TLS in transit; no plaintext in the document store	Compromise of the key-management component
Insider with a role	Per-request role check; approval gate; time window; every attempt logged	Misuse inside the permitted window
Privileged operator	GCM tag; digest anchored on the ledger; audit records mirrored on the ledger	Operator who also controls the private key
Recipient of a released copy	Per-recipient watermark; access log	Deterrence and attribution only, not prevention

3. System design

3.1 Architecture

ExamCrypt uses the layered structure shown in Fig. 1. Users reach a React web interface; every call passes through a Node.js and Express control layer that performs authentication, role checks and workflow control. That layer is the only component that talks to the four back-end services: the cryptography module, a PostgreSQL database hosted on Supabase, an Ethereum-compatible ledger (Ganache during development), and an encrypted document store that holds ciphertext only. Keeping the services behind a single control layer means a request cannot reach a paper or a key without passing the same checks.

**Fig. 1 - Layered architecture of ExamCrypt.**

3.2 Roles and access control

Four roles are defined. Permissions are granted to roles and checked on every request, and a user holds the minimum role needed for the task (Table 3). A request is evaluated afresh each time, regardless of whether the same user was authorised a moment earlier.

Table 3 - Roles and permitted operations.

Role	Permitted operations
Administrator	Manage users and roles; configure release schedules; view audit logs; verify document integrity
Examiner	Authenticate; upload a paper; set time-lock and watermark settings; track approval status

Role	Permitted operations
Approver	Review submitted papers; approve or reject; authorise controlled release
Invigilator	Access the released paper after the release time (watermarked copy)

3.3 Paper lifecycle

Fig. 2 shows the lifecycle. After the examiner authenticates and uploads a paper (PDF or DOCX), the system validates it, computes its SHA-256 digest, encrypts it, stores the ciphertext, and writes an audit record whose digest is submitted to the ledger. The paper is then placed in the approver's queue. A rejection is recorded and ends the workflow. An approval authorises release but does not open the paper: it stays locked until the scheduled time, when the system decrypts it for authorised users, applies a watermark, and logs the access. The digest is computed before encryption on purpose, so that it describes the paper the examiner actually submitted.

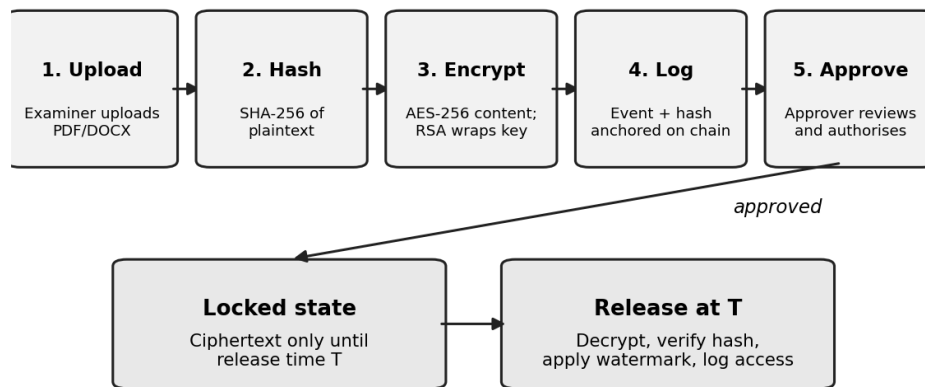


Fig. 2 - Secure paper lifecycle from upload to timed release.

3.4 Data model

Relational data lives in PostgreSQL. The users table is the central entity and is referenced by the other tables (Table 4). The blockchain reference stored with each audit row is what connects a database record to its entry on the ledger, so the two can be cross-checked in either direction.

Table 4 - Database tables and their purpose.

Table	Purpose
users	Identity, role and active status of each user (password stored as a hash)
documents	Paper metadata, ciphertext location, plaintext digest, status, uploader
encryption_records	Algorithms used, wrapped-key reference, digest value, who encrypted and when
document_access	Access grants per user and document, with granter and expiry
approvals	Approval requests and decisions, requester, approver and remarks
time_locks	Release time, expiry time and lock status for each document
watermarks	Watermark text and the user it was applied for
audit_logs	Action, details, data hash and the blockchain reference for each event

4. Algorithms

4.1 Encryption and key protection

For a paper M , the system draws a fresh 256-bit key K and a 96-bit nonce IV from a cryptographically secure generator and computes

$$C, \tau = \text{AES-256-GCM}(K, IV, M) \quad (1)$$

where C is the ciphertext and τ the authentication tag. The key is then wrapped under the institution's RSA public key pk :

$$EK = \text{RSA-OAEP}(pk, K) \quad (2)$$

The tuple (C, IV, τ, EK) is stored; K is discarded. A new key and nonce are generated for every paper, so the nonce never repeats under a given key. Because only the 32-byte key passes through RSA, the cost of the public-key step does not depend on the size of the paper.

4.2 Integrity verification

Before encryption the system computes

$$h = \text{SHA-256}(M) \quad (3)$$

and records h in the database and on the ledger. At every access, and after every transfer, the recovered plaintext M' is hashed and compared with the anchored value; a mismatch raises an integrity violation. The two checks are complementary. The GCM tag detects modification of the stored ciphertext. The anchored digest additionally detects the case in which an operator substitutes a different paper that has been properly encrypted under the same key, which passes the tag check but cannot match h . Section 7 tests both cases.

4.3 Controlled release

A request by user u for document d at time t is granted only if every condition below holds:

$$\text{Grant}(u, d, t) = \text{Auth}(u) \wedge \text{Perm}(\text{role}(u), \text{read}) \wedge \text{Approved}(d) \wedge \text{Trel} \leq t < \text{Texp} \quad (4)$$

where Trel and Texp are read from the `time_locks` table. If the request is granted, the paper is decrypted, verified against h , watermarked with the requester's identity and the access time, and delivered. In either case the outcome is written to the audit log; a denial is recorded as carefully as a grant, since repeated denied attempts are themselves useful evidence.

4.4 Audit anchoring

For each event e (upload, approval, decryption, denial and so on) the system builds a record containing the actor, the document, the action and a timestamp, computes its digest, and submits the digest with the document reference to the ledger. The transaction identifier is returned and stored as the blockchain reference in `audit_logs`. To audit an event, the stored record is re-hashed and compared with the on-chain value. Only digests are written to the chain, which keeps paper content and personal data off the ledger.

5. Implementation environment

The prototype is developed with the Agile model in four two-week sprints: requirements and architecture, encryption and authentication, ledger integration and approval workflow, and finally interface, testing and documentation. Table 5 lists the technologies used for each layer of the architecture in Fig. 1. The cryptographic operations rely on the built-in Node.js crypto module (Node.js v24.14.0) rather than hand-written primitives. Ganache, an Ethereum-compatible development network, serves as the ledger during development, and a permissioned platform such as Hyperledger Fabric 2.5.x is considered for deployment. The development machines run Windows 11 (64-bit). For deployment, a standard server with a quad-core class CPU, 8 GB of RAM and a 256 GB SSD is sufficient, and clients need only a browser.

Table 5 - Technologies used in ExamCrypt.

Layer	Technology	Role in the system
Front end	React.js 18.x, Tailwind CSS 3.x	Role-based web interface for login, upload, approval and release
Application layer	Node.js with Express.js	Authentication, role-based access control, workflow and document processing
Cryptography	Node.js crypto module (v24.14.0): AES-256, RSA, SHA-256	Encryption, key wrapping and hashing
Database	PostgreSQL 15+ via Supabase	Users, documents, approvals, time locks and audit records
Audit ledger	Ganache (Ethereum-compatible); Hyperledger Fabric 2.5.x for deployment	Immutable anchoring of digests and audit events
Document storage	Encrypted document store	Ciphertext of question papers only

Layer	Technology	Role in the system
Transport security	TLS	Protection of traffic between components
Platform	Windows 11 (64-bit); modern web browser on clients	Development environment and client access

6. Preliminary evaluation

6.1 Method

We measured the cryptographic path in isolation, using the Node.js crypto module (v24.14.0, OpenSSL-backed) on a Windows x64 laptop (AMD Ryzen 5 7520U, 16 GB RAM). Documents of 0.1, 1, 5, 10 and 25 MB were generated from random bytes; for AES and SHA-256 the running time depends on length rather than content, so random data is a fair stand-in for PDF and DOCX files of the same size. Each operation was run 15 times after two warm-up runs and the median is reported. The key-wrapping measurements use 50 runs. The ledger, database and network are deliberately excluded, so these figures show the cost of the cryptography itself and not the end-to-end response time.

6.2 Cost of hashing and encryption

Table 6 and Fig. 3 give the results. Times grow roughly in proportion to document size, as expected for a stream cipher mode and a hash function. Even the largest test file, at 25 MB, is processed in well under a tenth of a second. Typical question papers are far smaller, so the cryptographic cost is negligible next to network transfer and interface time.

Table 6 - Median processing time (ms) by document size.

Size (MB)	SHA-256	AES-256-GCM encrypt	AES-256-GCM decrypt	Hash verify
0.1	0.10	0.20	0.20	0.11
1	0.99	2.21	2.18	1.02
5	2.72	8.41	7.47	2.72
10	5.49	15.2	15.2	5.36
25	14.1	26.8	28.0	13.3

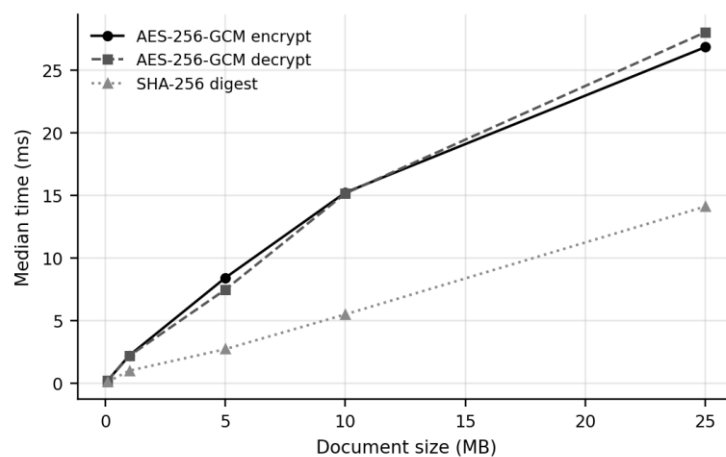


Fig. 3 - Processing time against document size.

6.4 Cost of key protection

Wrapping the 32-byte key with RSA-OAEP took 0.024 ms with a 2048-bit key and 0.043 ms with a 3072-bit key. Unwrapping, which uses the private key, took 0.51 ms and 1.50 ms respectively (Table 7). Key-pair generation is the only expensive step, at 54.5 ms and 62.1 ms for one run, but it happens

once per key pair and not once per paper. The private-key operation is the slower of the two and is performed once per release, so moving from 2048 to 3072 bits is affordable if the institution's policy asks for it.

Table 7 - RSA-OAEP key protection time (ms) for a 32-byte key.

Modulus (bits)	Wrap (public key)	Unwrap (private key)	Key generation (single run)
2048	0.024	0.51	54.5
3072	0.043	1.50	62.1

6.5 Tamper detection

To test integrity checking we used a 1 MB document and ran 1,000 trials for each of three cases. In the first, one randomly chosen bit of the stored ciphertext was flipped; decryption failed on the authentication tag in 1,000 of 1,000 trials. In the second, one random bit of the plaintext was flipped; the recomputed SHA-256 digest differed from the reference in 1,000 of 1,000 trials. In the third, a different paper was properly encrypted under the same key and substituted for the approved one. This passed the tag check in 1,000 of 1,000 trials, as it should, since the ciphertext is internally consistent, and the digest comparison then rejected it in 1,000 of 1,000 trials. The third case shows why the digest is anchored outside the database and is not left to the cipher alone. These results confirm that the primitives are used correctly. They are not a substitute for testing the complete system, and we do not present them as such.

6.6 Evaluation still to be completed

Four parameters identified in the project plan are not measured here: ledger transaction latency, correctness of role enforcement across all four roles, enforcement of the approval and time conditions at the boundaries of the release window, and detection of altered audit records. They require the complete integrated system, which is being built in the later sprints, and will be reported separately.

7. Discussion and limitations

The design shifts trust rather than removing it. Anchoring digests and audit records on a ledger means an operator can no longer rewrite history unnoticed, but an operator who also controls the RSA private key could still decrypt a paper outside the workflow. In that case the system would not prevent the act, although a decryption that bypasses the application leaves no matching ledger entry, and an audit that looks for such gaps would reveal it. Holding the private key in a hardware security module, and splitting it among several custodians so that release needs a threshold of them, would reduce this exposure further, and is the most important item of future work.

Second, the time gate is a policy enforced by the application and by key custody, not a cryptographic time lock; it is only as strong as the control layer that evaluates it. We chose it deliberately, for the reason given in Section 2, but it means that the release moment is trusted to the server clock. Third, watermarking identifies the source of a leak after the fact. It cannot stop a recipient from photographing a screen. Fourth, the benchmarks in this paper cover the cryptographic path on one machine and are indicative rather than definitive; ledger latency in particular is expected to be much larger than the figures above and may set the real response time of audit-bound operations.

8. Conclusion and future work

We presented ExamCrypt, a design for managing examination papers under cryptographic control from upload to timed release. It combines per-document AES-256-GCM encryption with RSA-OAEP key wrapping, SHA-256 digests anchored on a ledger, role-checked and approval-gated release, and per-recipient watermarking. Preliminary measurements show that the cryptographic cost is a few milliseconds for typical documents, and that tampering with ciphertext, plaintext or the paper itself was detected in every trial.

Next steps are the integrated evaluation described above, hardware-backed and threshold key custody, extension to several institutions, a mobile client for approvers and invigilators, forensic watermarking with anomaly detection on audit logs, and, further out, support for post-examination material such as answer sheets and results.

Acknowledgements

The authors thank their internal guide, Dr. Sunil Chavan, for guidance throughout this work, and the Department of Computer Engineering for the facilities provided.

References

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., et al. (2018). Hyperledger Fabric: A distributed operating system for permissioned blockchains. *Proceedings of the Thirteenth EuroSys Conference*, Article 30.

- Bellare, M., & Rogaway, P. (1994). Optimal asymmetric encryption. In *Advances in Cryptology – EUROCRYPT '94* (pp. 92–111). Springer.
- Dworkin, M. (2007). *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC* (NIST Special Publication 800-38D). National Institute of Standards and Technology.
- Leela, K., & Vinod, S. (2018). Implementation of digital signature using hybrid cryptosystem. *International Journal of Engineering and Technology*.
- Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system*. <https://bitcoin.org/bitcoin.pdf>
- National Institute of Standards and Technology. (2001). *Advanced Encryption Standard (AES)* (FIPS PUB 197).
- National Institute of Standards and Technology. (2015). *Secure Hash Standard (SHS)* (FIPS PUB 180-4).
- Ramachandran, A., & Kantarcioglu, M. (2017). Using blockchain and smart contracts for secure data provenance management. *arXiv preprint arXiv:1709.10000*.
- Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2), 120–126.
- Rivest, R. L., Shamir, A., & Wagner, D. A. (1996). *Time-lock puzzles and timed-release crypto* (MIT/LCS/TR-684). Massachusetts Institute of Technology.
- Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). *Zero trust architecture* (NIST Special Publication 800-207). National Institute of Standards and Technology.
- Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2), 38–47.
- Wood, G. (2014). *Ethereum: A secure decentralised generalised transaction ledger* (Yellow Paper).
- Zyskind, G., Nathan, O., & Pentland, A. (2015). Decentralizing privacy: Using blockchain to protect personal data. *Proceedings of the IEEE Security and Privacy Workshops*, 180–184.