Accountable Decryption made Formal and Practical

Rujia Li, Yuanzhao Li, Qin Wang, Sisi Duan, Qi Wang, Mark Ryan

Tsinghua University, China
University of New South Wales, Australia
University of Birmingham, United Kingdom
Southern University of Science and Technology, China

November 15, 2023

Preventive Strategies

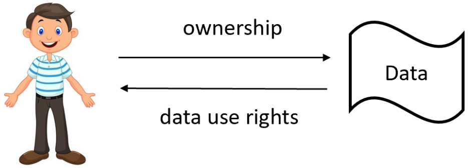
Modern cryptographic systems rely on preventive strategies:

- Passwords, access control
- Access control
- Authentication protocols

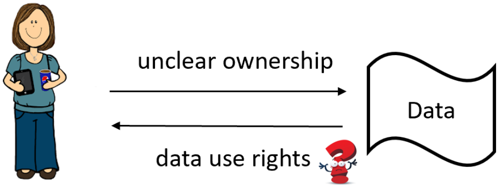
However, as computer systems become more complex and larger in scale, preventive strategies alone are not enough.

Data Use Rights

The data owner has data use rights.



Others do not have data use rights ???



The Needs for Accountability

In certain break-glass scenarios, users might need to bypass these strategies to access important information. This is where accountability plays a vital role:

- Complements preventive strategies.
- Identifies and penalizes misuse after data access.
- Ensures responsible use of sensitive data.

Make Decryption Accountable

- Detection of unauthorized access
  allowing the encryptor to audit the decryption process
- Deterrents of illegal behavior
  any violation of the access control to data can be caught and punished
- Regulatory compliance
  verifiable record on decryption is provided for checking its compliance with the pre-defined policies
- Key leakage awareness
  users will be alerted of a potential leakage of keys (ciphertexts decrypted without permission)

Contents

- Definition
- Scheme
- Implementation
- Evaluation
- Applications

Accountability Definition

Definition (1996, On linkage)

An accountable system associates states and actions with identities and provides primitives for actors to validate the states and actions of their peers, such that cheating or misbehavior becomes detectable, provable, and undeniable by the perpetrator[1]*.*

Definition (2000, On detection)

Accountability has the following features: (a) reliable evidence: It indicates the delivery to a principal of evidence that is later presented to the judge. The evidence can be validated and achieves fairness (i.e., that one protocol participant gets evidence if and only if the other one does) and non-repudiation; (b) enhanced misconduct detection: A system should provide a means to directly detect and expose misbehavior by its participants, or, enable a principal to prove to the judge any detected fraud[2]*.*

[1] Kailar, Rajashekar. "Accountability in electronic commerce protocols." IEEE Transactions on Software Engineering 22, no. 5 (1996): 313-328.

[2] Buldas, Ahto, Helger Lipmaa, and Berry Schoenmakers. "Optimally efficient accountable time-stamping." International workshop on public key cryptography, pp. 293-305. Springer Berlin Heidelberg, 2000.

Accountability Definition

Definition (2005, On punishment)

Accountability in a computing system implies the following properties: (a) awareness of policy violation: Some actors have the right to hold other actors to a set of standards and judge whether they have fulfilled their responsibilities in light of these standards. (b) posterior penalty: An entity is accountable with respect to some policy (or accountable for obeying the policy). Whenever the entity violates the policy, with some non-negligible probability, the entity will be punished[3].

Definition (2010, detection and punishment)

A system is accountable if (a) faults can be reliably detected, (b) each fault can be undeniably linked to at least one faulty node, and (c) the faulty entities will be properly sanctioned[4].

[3] Feigenbaum, Joan, Aaron D. Jaggard, and Rebecca N. Wright. "Towards a formal model of accountability." In Proceedings of the 2011 New Security Paradigms Workshop, pp. 45-56. 2011.

[4] Haeberlen, Andreas. "A case for the accountable cloud." ACM SIGOPS Operating Systems Review 44, no. 2 (2010): 52-57.

General Principles of Accountability

- Linkage identities: Actions linked to entities performing them.
- Reliable evidence: Records of actions preventing secret omissions or falsifications.
- Policy compliance: Evidence inspection for faults.
- Detection: Fault alerts verifiable by third parties.
- Punishment: Sanctions for misconduct.

Our Definition of Accountable Decryption

Our approach to accountable decryption focuses on responsibility and fairness:

- *Trace*, *identify*, and *punish* malicious decryptors.
- Feedback and penalties based on evidence.

Entities and Syntax

Accountable decryption involves multiple entities:

- **Encryptor** (E): Creates ciphertexts and policies.
- **Decryptor** (D): Performs decryption, generating plaintext and evidence.
- **Judge** (J): Detects misbehavior and imposes penalties.

The key operations include:

- **Encryption**: E generates ciphertext and policies.
- **Decryption**: D decrypts under specific conditions, producing evidence.
- **Check**: J ensures actions comply with policies.
- **Reaction**: J penalizes non-compliant actions.

Entities and Syntax

- **Encryption**. $(ct, \mathcal{P}) \leftarrow \mathsf{Enc}(m)$: An encryptor E executes this algorithm to generate a ciphertext ct, and policies $\mathcal{P}$. Here, $\mathcal{P}$ dictates what are legal actions.
- **Decryption**. $(m, \pi) \xleftarrow{\widetilde{e}} \mathsf{Dec}(key, ct)$: A decryptor D executes this algorithm under designated environment denoted as $\widetilde{e}$. $\widetilde{e}$ captures critical aspects of the event, encompassing precise timing, unfolding sequence, and the identities of the participating entities. Ideally, π faithfully reports $\widetilde{e}$.
- **Check**. $tag \leftarrow \mathsf{Check}(\pi, ct, \mathcal{P})$: A judge J executes this algorithm to scrutinize the actions of the decryptor, ensuring compliance with the predefined policies. Here, *true* indicates the decryptor's action is aligned with policies.
- **Reaction**. $\perp \leftarrow \mathsf{React}(tag, \mathcal{P})$: A judge J imposes penalties against the decryptor in case of non-compliance.

Definition 1: Accountability of Decryption

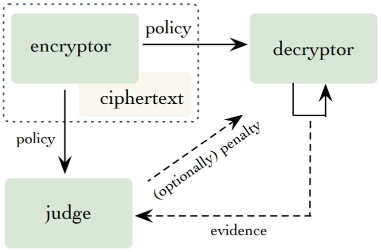
Definition (Accountability of decryption, ADec)

A system achieves ADec *if the following conditions hold:*

(i) *non-repudiation: for any execution of* $\mathsf{Dec}(key, ct)$ *on* $\widetilde{e} \nprec \mathcal{P}$, *there exists a negligible function* negl *that makes* $\mathsf{Check}(\pi, \mathcal{P})$ *output true, namely,* $\Pr[true = \mathsf{Check}(\pi, \mathcal{P})] \leq \mathsf{negl}(\lambda)$.

(ii) *non-frameability: for any execution of* $\mathsf{Dec}(key, ct)$ *on* $\widetilde{e} \prec \mathcal{P}$, *there exists a negligible function* negl *that makes* $\mathsf{Check}(\pi, \mathcal{P})$ *output false, namely,* $\Pr[false = \mathsf{Check}(\pi, \mathcal{P})] \leq \mathsf{negl}(\lambda)$.

This ensures that malicious decryptors are identified and penalized, while honest ones are safeguarded.



Definition 2: Accountability with Trustworthy Trustee

Definition (Accountability of decryption with **trustworthy** trustee, ADec-TS)

The system achieves ADec-TS *if the following conditions hold:*

(i-ii) *the same as those in Definition 2.*

(iii) *key-privacy: for any execution of* $\mathsf{Dec}(key, ct)$*, the probability for* TS *to leak the key is negligible.*

(iv) *evidence-authenticity: for any execution of* $\mathsf{Dec}(key, ct)$ *with* $\widetilde{e}$*, the probability for* TS *to output a forged* π' *is negligible, where* $\pi \neq \pi'$*, and* $\mathsf{Check}(\pi, ct, \mathcal{P}) = \mathsf{Check}(\pi', ct, \mathcal{P})$*.*

(v) *evidence-completeness: for any execution of* $\mathsf{Dec}(key, ct)$*, the probability for* TS *to fail to output evidence is negligible, namely* $\mathsf{Pr}[m, \perp = \mathsf{Dec}(key, ct)] \leq \mathsf{negl}(\lambda)$*, where* $\perp$ *signifies the absence of evidence being produced.*

This definition adds layers of security and trust, ensuring the trustee's role enhances accountability.

Definition 3: Accountability with Untrusted Trustee

Definition (Accountability of decryption with **untrusted** trustee, ADec-uTS)

The system achieves ADec-uTS *if the following conditions hold:*

(i) *non-repudiation: for any execution of* $\mathsf{Dec}(key, ct)$ *on* $\widetilde{e} \nprec \mathcal{P}$*, there exists a negligible function* negl *that makes* $\mathsf{Check}(\pi, \mathcal{P})$ *output true, namely,* $\Pr[true = \mathsf{Check}(\pi, \mathcal{P})] \leq \mathsf{negl}(\lambda)$.

(ii) *non-frameability: for any execution of* $\mathsf{Dec}(key, ct)$ *on* $\widetilde{e} \prec \mathcal{P}$*, there exists a negligible function* negl *that makes* $\mathsf{Check}(\pi, \mathcal{P})$ *output false, namely,* $\Pr[false = \mathsf{Check}(\pi, \mathcal{P})] \leq \mathsf{negl}(\lambda)$.

(vi) *leakage-resistance: Even if* TS *is compromised, the probability for* TS *to obtain* D*'s private key is negligible.*

(vii) *compromise-awareness : If* TS *fails to meet Condition-(iv) or Condition-(v) as specified in Definition 6, it exposes itself to the risk of detection. Alternatively, when* TS *misbehaved, the probability of the victim user's (i.e., encryptor) being* ***unaware*** *of* TS*'s misbehavior is negligible.*

This definition ensures the system's security and integrity, even in the worst-case scenario of a compromised trustee.

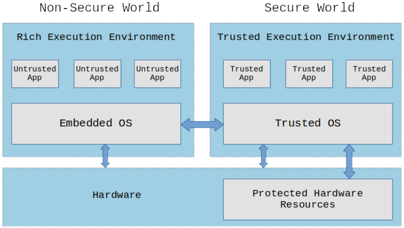
Contents

- Definition
- Scheme
- Implementation
- Evaluation
- Applications

Trusted Execution Environment

TEE is a secure area within the main processor that operates as an isolated kernel, ensuring the confidentiality and integrity of sensitive data and computations[5].

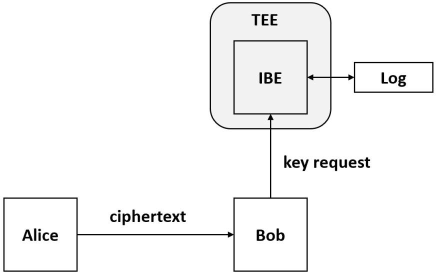
- Isolated execution:
  Protected execution zone
- Ability to convince verifiers:
  (Remote/Local) Attestation
- Protected storage:
  Sealing



[5] Li, Rujia, Qin Wang, Qi Wang, David Galindo, and Mark Ryan. "SoK: TEE-Assisted Confidential Smart Contract." Proceedings on Privacy Enhancing Technologies 3 (2022): 711-731.

Naive Solution

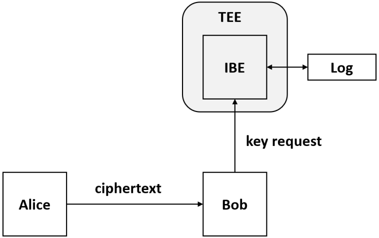
- Running a key manager (e.g., identity-based encryption scheme[6]) inside TEE
- Forcing TEE to generate a piece of evidence for each key request



[6] Boneh, Dan, and Matt Franklin. "Identity-based encryption from the Weil pairing." In Annual international cryptology conference, pp. 213-229. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

Workflow

- An encryptor encrypts a message and sends a ciphertext ct and $\mathcal{P}$ to a decryptor.
- A decryptor submits a decryption request to TEE-based TS with a signature to prove the identity.
- TEE retrieves the decryption key while generating evidence about the key request.
- The decryptor accesses the secret by decrypting ct.
- J checks the evidence and imposes penalties against malicious decryptors.



Technical Challenges

- **Challenge I:** How to **protect** the decryption keys under compromised TEEs?
- **Challenge II:** How to **detect** the compromised TEE?

Mitigating Challenge I: Protect

$\mathsf{IBE.KGen}_{PKG}(msk, ID, C)$

$r', t_1 \xleftarrow{\$} \mathbb{Z}_p^\star;$

$d_1' \leftarrow (Y \cdot C \cdot h^{t_1})^{1/x} \cdot H_{\mathbb{Z}}(ID)^{r'}$

$d_2' \leftarrow X^{r'}$

$d_3' \leftarrow t_1$

$pkey \leftarrow (d_1', d_2', d_3')$

return $pkey$

$\mathsf{IBE.KGen}_{D2}(mpk, ID, pkey)$

$r'' \xleftarrow{\$} \mathbb{Z}_p^\star$

$r \leftarrow r' + r''$

$d_1 \leftarrow \frac{d_1'}{g^\theta} \cdot H_{\mathbb{Z}}(ID)^{r''}$

$d_2 \leftarrow d_2' \cdot X^{r''}$

$d_3 \leftarrow d_3' + t_1$

$key \leftarrow (d_1, d_2, d_3)$

checks $e(d_1, X) = e(Y, g) \cdot e(h, g)^{d_3} \cdot e(H_{\mathbb{Z}}(ID), d_2)$

return key

Key-splitting mechanism

- Using a commitment scheme to hide a random number
(used to generate the full key).
- TEE only stores a partial key while the user holds another partial one.
- Even if an attacker accesses the key inside TEEs, it cannot obtain a full decryption key.

Mitigating Challenge II: Detect

For identifying potential compromises of TS, we introduce two sub-algorithms:

- **Deterministic detection algorithm** infers a compromised state through a challenge-response mechanism.
- **Probabilistic detection algorithm** identifies potential compromises by comparing the final keys issued by D with those issued by TS.

Algorithm 1 The *detection* algorithm

wrongdoing $\in \{true, false\}$
- - - - - - - - - - **deterministic detection** - - - - - - - - - -
upon receiving π and $pkey$ from TS
  $key = \mathsf{IBE.KGen}_{D2}(mpk, ID, pkey)$
  if $\pi \notin \mathcal{LOG} \wedge \mathsf{Dec}(key, ct) \neq \perp$ **then**
    wrongdoing $= true$ ▷ TS must have forged π
upon receiving π and $pkey$ from TS
  if $\pi \in \mathcal{LOG} \wedge \mathsf{IBE.KGen}_{D2}(mpk, ID, pkey) = \perp$ **then**
    wrongdoing $= true$ ▷ TS must have forged $pkey$
upon receiving $pkey$ from TS
  $key = \mathsf{IBE.KGen}_{D2}(mpk, ID, pkey)$
  if find_evidence$(\mathcal{LOG}) = \varnothing \wedge \mathsf{Dec}(key, ct) \neq \perp$ **then**
    wrongdoing $= true$ ▷ TS must have suppressed π
upon receiving π from TS
  if find_key$(\pi) = \varnothing$ **then**
    wrongdoing $= true$ ▷ TS must have suppressed $pkey$
- - - - - - - - - - **probabilistic detection** - - - - - - - - - -
upon finding key'
  if $key \neq key' \wedge \mathsf{Dec}(key', ct) \neq \perp$ **then**
    wrongdoing $= true$

Mitigating Challenge II: Detect

Deterministic detection algorithm

- TS forged π
- TS forged $pkey$
- TS suppressed π
- TS suppressed $pkey$

Probabilistic detection algorithm

- TS hides the new key and evidence

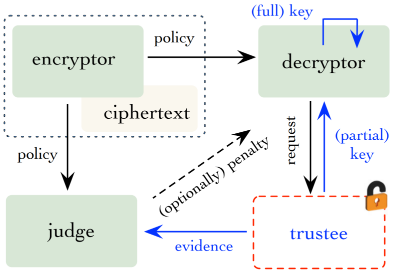
Algorithm 1 The *detection* algorithm

wrongdoing $\in \{true, false\}$

– – – – – – – – – – **deterministic detection** – – – – – – – – – –
upon receiving π and $pkey$ from TS
  key = IBE.KGen$_{D2}(mpk, ID, pkey)$
  if $\pi \notin \mathcal{LOG} \wedge$ Dec$(key, ct) \neq \perp$ **then**
    wrongdoing $= true$ ▷ TS must have forged π
upon receiving π and $pkey$ from TS
  if $\pi \in \mathcal{LOG} \wedge$ IBE.KGen$_{D2}(mpk, ID, pkey) = \perp$ **then**
    wrongdoing $= true$ ▷ TS must have forged $pkey$
upon receiving $pkey$ from TS
  key = IBE.KGen$_{D2}(mpk, ID, pkey)$
  if find_evidence$(\mathcal{LOG}) = \varnothing \wedge$ Dec$(key, ct) \neq \perp$ **then**
    wrongdoing $= true$ ▷ TS must have suppressed π
upon receiving π from TS
  if find_key$(\pi) = \varnothing$ **then**
    wrongdoing $= true$ ▷ TS must have suppressed $pkey$
– – – – – – – – – – **probabilistic detection** – – – – – – – – – –
upon finding key'
  if $key \neq key' \wedge$ Dec$(key', ct) \neq \perp$ **then**
    wrongdoing $= true$

Final Scheme

- E encrypts a message and sends the corresponding ciphertext ct and $\mathcal{P}$ to D.
- for decrypting ct, D must send a key request with a commitment for his random number to TS.
- TS generates decryption keys and updates the evidence π of key extraction.
- J traces log to find the misbehavior of decryption and imposes penalties against the decryptor.
- The inspectors (i.e., E, D, J) identify and prove the guilty of dishonest/compromised TEE who suppressed the evidence/key and provided the forged evidence/key.



Detailed Protocol

$\mathsf{Portex.Setup}(\lambda)$

PKG *runs* $pms_{hw} \leftarrow \mathsf{HW.Setup}(\lambda)$
$hdl_{GE} \leftarrow \mathsf{HW.Load}(pms_{hw}, Q_{GE})$
$mpk, vk_{GE} \leftarrow$ $\mathsf{HW.Run\&Quote}(hdl_{GE}, (\text{“}init\text{”}, \lambda))$
- $H \leftarrow 0$
- $mpk, msk \leftarrow \mathsf{IBE.Setup}(\lambda)$
- $vk_{GE}, sk_{GE} \leftarrow \mathsf{S.KGen}(\lambda)$
- **return** mpk, vk_{GE}

LM *runs* $vk_{LM}, sk_{LM} \leftarrow \mathsf{MT.Init}(\lambda)$
CLIENT *runs* $pk_{cli}^{enc}, sk_{cli}^{enc} \leftarrow \mathsf{PKE.KGen}(\lambda)$
$vk_{cli}^{sig}, sk_{cli}^{sig} \leftarrow \mathsf{S.KGen}(\lambda)$

$\mathsf{Portex.LogTrace}(\alpha, \pi)$

$(ID^\star, SN^\star, \tau^\star) \xleftarrow{parse} \alpha$
$(N, H', H, \rho, \varepsilon) \xleftarrow{parse} \pi$
$(ID, SN, \tau, \sigma_{cli}) \xleftarrow{parse} N$
return $((ID, SN, \tau) == (ID^\star, SN^\star, \tau^\star))$
$\wedge\ (true == \mathsf{MT.Verify}(\pi))$
$\wedge\ (true == \mathsf{S.Verify}(vk_{cli}^{sig}, \sigma_{cli}, ID|SN))$

$\mathsf{Portex.TEEInspect}(key, D)$

$(d_1, d_2, d_3) \xleftarrow{parse} key$
$(D_1, D_2, D_3) \xleftarrow{parse} D$
return $d_3 \neq D_3$

$\mathsf{Portex.Enc}(ID, m)$

$SN \xleftarrow{\$} \mathbb{Z}_m^\star$
$ct \leftarrow \mathsf{IBE.Enc}(mpk, ID|SN, m)$

$\mathsf{Portex.KReq}(ID, SN)$

CLIENT *runs* $C \leftarrow \mathsf{IBE.KGen}_{D1}(mpk)$
$\sigma_{cli} \leftarrow \mathsf{S.Sign}(sk_{cli}^{sig}, ID|SN)$
CLIENT *sends* $(ID, SN, C, \sigma_{cli})$ *to* LM.
$if\,(true \neq \mathsf{S.Verify}(vk_{cli}^{sig}, \sigma_{cli}, ID|SN)), abort$
LM *runs* $\pi \leftarrow \mathsf{MT.Insert}(ID, SN, \tau, \sigma_{cli})$
$ir = (\pi, C), \sigma_{ir} \leftarrow \mathsf{S.Sign}(sk_{LM}, ir)$
LM *sends* (ir, σ_{ir}) *to* PKG
$quote \leftarrow$ $\mathsf{HW.Run\&Quote}(hdl_{GE}, (\text{“}kreq\text{”}, ir, \sigma_{ir}))$
- $if\,(true \neq \mathsf{S.Verify}(vk_{LM}, \sigma_{ir}, ir)), abort$
- $if\,(H \neq H_{old}), abort$
- $(\pi, C) \xleftarrow{parse} ir$
- $(N, H_{new}, H_{old}, \rho, \varepsilon) \xleftarrow{parse} \pi$
- $(ID, SN, \tau, \sigma_{cli}) \xleftarrow{parse} N$
- $if\,(false = \mathsf{MT.Verify}(\pi)), abort$
- $H \leftarrow H_{new}$
- $pkey \leftarrow \mathsf{IBE.KGen}_{PKG}(msk, ID|SN, C)$
- $ct_{pkey} \leftarrow \mathsf{PKE.Enc}(pk_{cli}^{enc}, pkey)$
- $\sigma_{pkey} \leftarrow \mathsf{S.Sign}(sk_{GE}, ct_{pkey})$
- **return** ct_{pkey}, σ_{pkey}

CLIENT *receives quote*
$(md_{hdl}, tag_Q, in, out(ct_{pkey}, \sigma_{pkey}), \sigma) \xleftarrow{parse} quote$
$if\,(tag_Q \neq tag_{GE}), abort$
$if\,(false = \mathsf{HW.VerifyQuote}(pms_{hw}, quote)), abort$
$if\,(false \leftarrow \mathsf{S.Verify}(vk_{GE}, \sigma_{pkey}, ct_{pkey})), abort$
$pkey \leftarrow \mathsf{PKE.Dec}(sk_{cli}^{enc}, ct_{pkey})$
$key \leftarrow \mathsf{IBE.KGen}_{D2}(mpk, ID|SN, pkey)$

$\mathsf{Portex.Dec}(key, ct)$

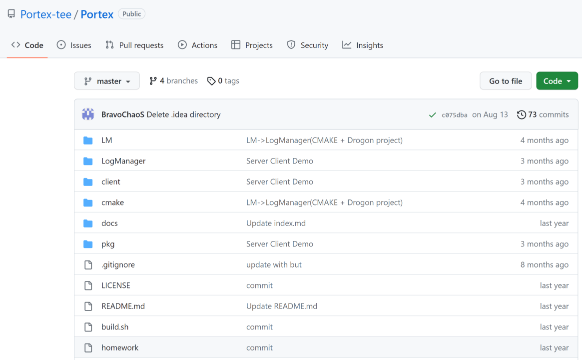
- Partial key created by users
- Partial key generated inside TEE
- Evidence generated inside TEE

Contents

- Definition
- Scheme
- Implementation
- Evaluation
- Applications

Implementation

- Intel SGX SDK (v2.14):
  Developing SGX applications
- GMP (v6.2.1):
  The GNU Multiple Precision
- PBC (v0.5.14):
  Pairing-Based Cryptography
- Merklecpp (v1.0.0):
  A simple Merkle tree library
- OpenSSL (v1.1.1u):
  Cryptography and SSL/TLS Toolkit
- OpenSSL (v2.14):
  Intel Software Guard Extensions SSL
- Drogon (v1.8.3):
  HTTP application framework



Demonstration

A-Dec

Home How A-Dec works Download Service Github

Accountable Decryption

For decryptor: Using trusted hardware to force each decryption to generate publicly verifiable logs, ensuring accountability.

For trusted hardware: Inspecting the TEE's outputs, aiming to reduce the risk of the compromised TEE.

Applications

Weblink

http://a-decrypt.com/

QR code

Contents

- Definition
- Scheme
- Implementation
- Evaluation
- Applications

Performance and Scalability Evaluation

We assess the performance of Portex focusing on two aspects:

- **Decryption performance**:
  misbehavior-checking & compromised TEE detection.
- **Scalability**:
  performance with an increasing number of decryptors.

Experimental setup:

- Hardware: 3.5GHz Intel Xeon CPU (Ice Lake).
- Security parameter: $\lambda = 512$.
- Pairing: Symmetric bilinear pairing on curve $y^2 = x^3 + x$.
- Methodology: Averaging over 1,000 repetitions.

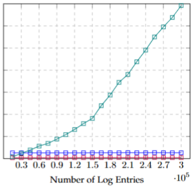
Evaluation Results

Performance with Static Parameters:

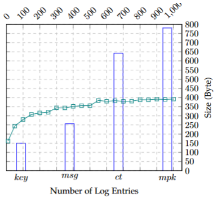
- Scenario: 1000 decryptors, each performing one decryption.
- Decryption time: Full process within 10ms.
  - *Key request*: 1.31ms, 3.15ms, 2.56ms for sub-algorithms.
  - *Evidence generation and verification*: 1.69ms and 6.05ms.
  - *Ciphertext decryption*: Less than 1ms (Dec.Setup: 0.1ms, IBE.Dec: 0.6ms).
- Tracing malicious decryption: Max 0.002ms.
- Detecting forgeries: 0.02ms for evidence, 0.01ms for keys.

Evaluation Results

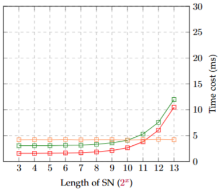
Results



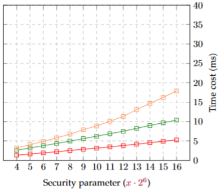
(a) Evidence verification (purple), generation (blue), and tracing (teal)



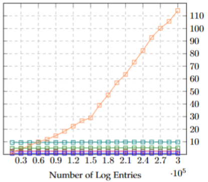
(b) The size of evidence (teal) and *key*, *ct*, *msg*, *mpk* (blue)



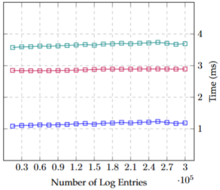
(c) KGen_{D1} (red), KGen_{PKG} (green), and KGen_{D2} (yellow) in algorithm IBE



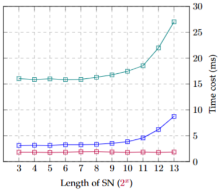
(d) KGen_{D1} (red), KGen_{PKG} (green), and KGen_{D2} (yellow) in algorithm IBE



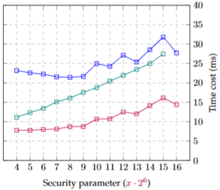
(e) Encryption (purple), Decryption (blue), Key Request by D (teal), by LM (yellow), and by PKG (green)



(f) Deterministic detection of Forged evidence (purple), of Forged key (teal), and probabilistic detection (blue)



(g) Encryption (blue), Key Request (teal), Decryption (purple)



(h) Encryption (blue), Key Request (teal), Decryption (purple)

Contents

- Definition
- Scheme
- Implementation
- Evaluation
- Applications

Applications

Accountable warrant execution: Making law enforcement officers accountable for accessing sensitive information.

Accountable ePHI: Helping electronically protected health information to find unauthorized access and potential breaches.

Accountable location access: Any access to the recipient's info is auditable, ensuring accountability and preventing misuse.

Access Control, Traceability, and Accountability

Access control	Governing who can access specific resources, defining user permissions, privileges, and restrictions.	A nurse may access patient records in her department but not in other departments.
Traceability	Providing a documented trail of how the data has been created, modified, accessed, or transferred.	A nurse accesses a patient's record in her department. The system records the nurse's name, time of access, and the reason for access.
Accountability	Checking the documented trail to make users accountable for any misuse or violations.	A nurse has improperly accessed a patient's records. Accountability requires she to provide valid reasons; otherwise, she will be sued.

Conclusion

- **Novel definitions**: Introduction of a new set of definitions specifically tailored for accountable decryption, aiming to encompass all potential scenarios and limitations.
- **Scheme construction**: Development of a scheme that aligns with these definitions, utilizing trusted hardware to ensure reliability and security.
- **Prototype and evaluation**: Implementation of a prototype demonstrating the practicability and efficiency of our approach, backed by thorough evaluations.

Thanks