Um esquema baseado em blockchain por Proof-of-Download para gerenciamento de direitos digitais e detecção de traidores

João Tito do Nascimento Silva
Felipe Z. da N. Costa
João Gondim

Departamento de Ciência da Computação - UnB
CIn - Centro de Informática - UFPE
Departamento de Engenharia Elétrica - UnB

Motivação

Digital Rights Management

O uso de tecnologia para gerenciar acesso a material protegido por direitos autorais e licenciamento

Traitor Detection

Técnicas que buscam fornecer evidências de que um certo ativo digital está sendo utilizado de forma *ilegal*, ou seja, sem o devido licenciamento.

Problema(s)

Mecanismos existentes

- Gestão centralizada - confiança em um terceiro
- Técnicas baseadas na gestão de IPs e redes dificultam uma verificação e utilização de licenças *offline* e exigem centralização.
- Pouca flexibilidade para utilização em diferentes indústrias
- Outras propostas ignoram a confidencialidade dos ativos

Desafio(s)

Atores e modelo de ameaça

Clientes x Provedores

- O provedor é dono de um ativo, e deseja receber um pagamento, enquanto o cliente quer ter acesso a esse ativo, sendo que o ativo é fornecido por meio de um *download* de arquivo
- Ambos podem agir de forma maliciosa

Desafio(s)

Atores e modelo ameaças

Verificadores

- Verificam pagamentos e downloads de forma decentralizada - modelo *honest-but-curious*
- Como conciliar confidencialidade dos ativos com verificadores de *download*?
- Necessidade de um protocolo com propriedade de *zero-knowledge*

Solução Proposta

Verificação de *downloads* com *Provable Data Provisioning* (PDPr)

- Ao final da execução, deseja-se:
  - O cliente tem acesso ao ativo
  - O provedor emitiu uma prova verificável de que o cliente pode acessar o ativo
  - O verificador deve ser capaz de verificar a prova mesmo que não tenha acesso ao ativo
- Zero-knowledge - não deve revelar nenhuma nova informação com suas execuções
- Inspirado no *Proof-of-Download* (PoDl) [1]

Solução Proposta

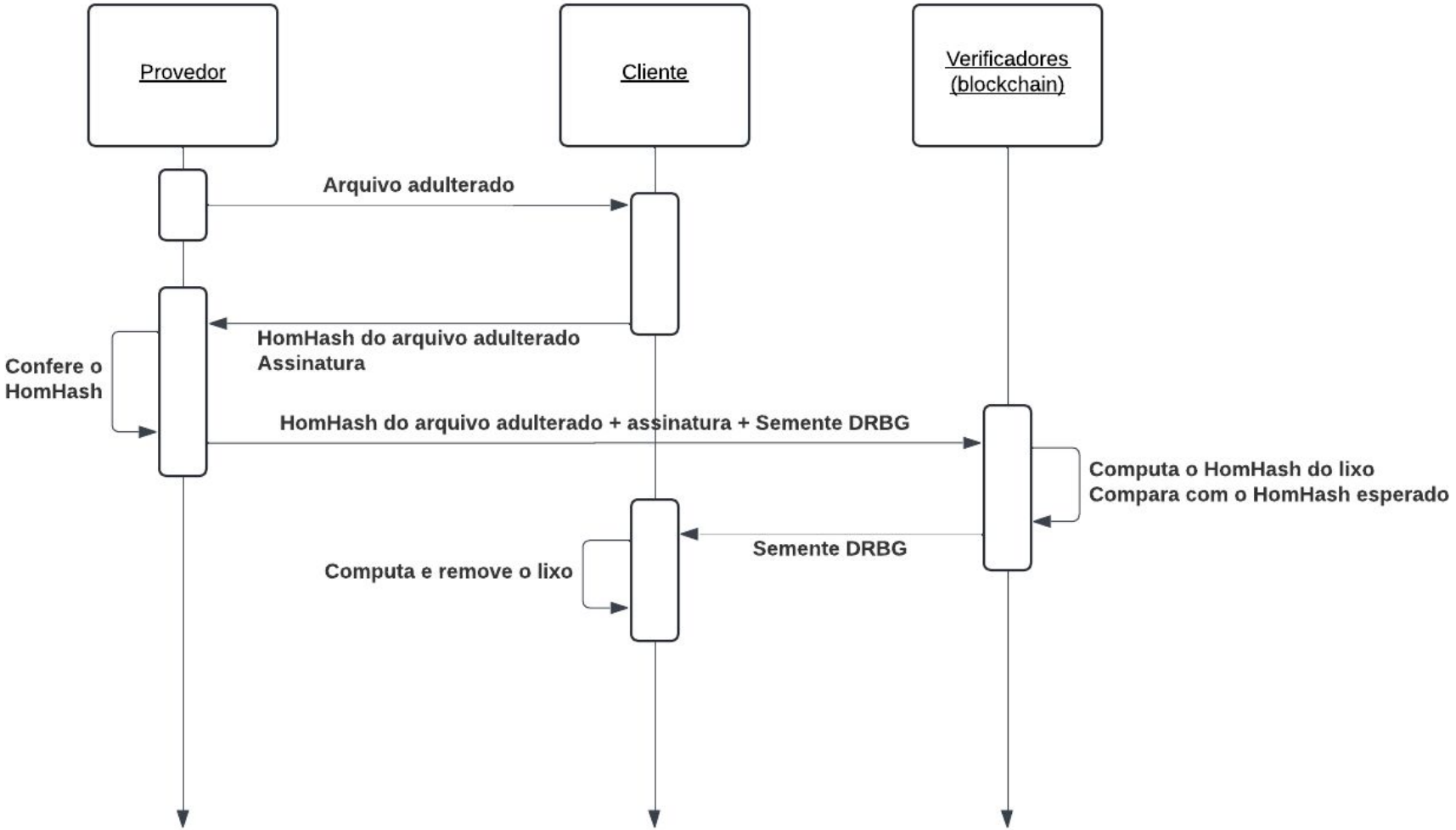
PDPr com *hash* homomórfico

- Hash homomórfico

$$HomHash(T) = HomHash(F) + HomHash(N)$$

$$HomHash(N) \stackrel{?}{=} HomHash(T) - HomHash(F)$$

- HomHash por incrementalidade - LtHash [2]


Provedor
Cliente
Verificadores (blockchain)
Arquivo adulterado
HomHash do arquivo adulterado
Assinatura
Confere o HomHash
HomHash do arquivo adulterado + assinatura + Semente DRBG
Computa o HomHash do lixo
Compara com o HomHash esperado
Semente DRBG
Computa e remove o lixo

Solução Proposta

Problemas do PDPr com *hash* homomórfico

- Requer quantidade grande de memória na verificação
- Inviável para um *smart contract*
- Sujeito a um ataque com múltiplos arquivos adulterados
  - Obter várias cópias, usando identidades diferentes
  - Intersectar as cópias
- Precisamos de um esquema de cifração mais robusto!

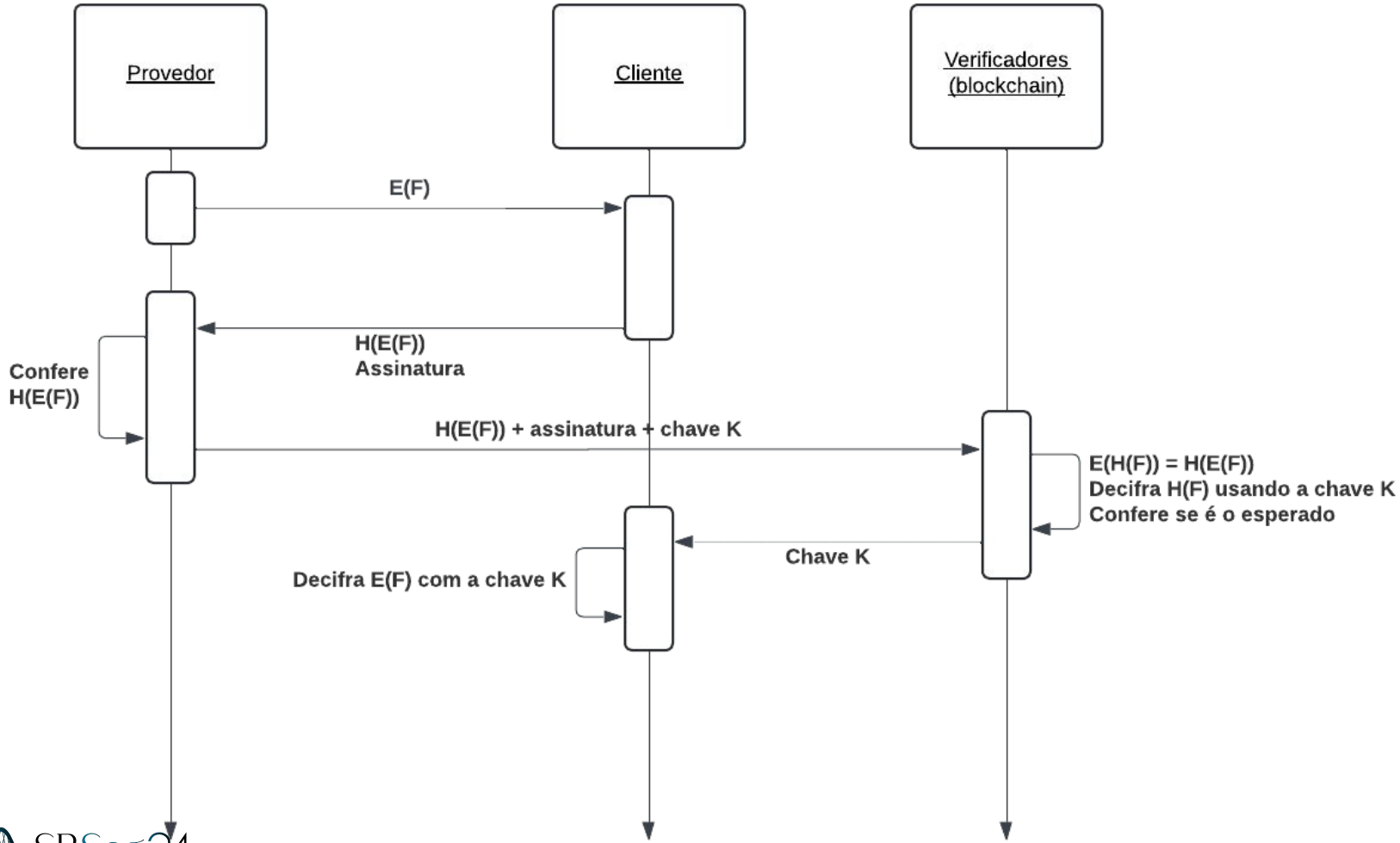
Solução Proposta

PDPr com criptografia homomórfica

- Obter um par de hash criptográfico e cifra homomórfica tais que

$$H(E_K(F)) = E_K(H(F))$$

- O provedor fornece E(F), e o cliente retorna H(E(F)) com assinatura
- Como prova, o provedor fornece a assinatura com a chave K publicamente


Provedor
Cliente
Verificadores (blockchain)
E(F)
H(E(F))
Assinatura
Confere H(E(F))
H(E(F)) + assinatura + chave K
E(H(F)) = H(E(F))
Decifra H(F) usando a chave K
Confere se é o esperado
Chave K
Decifra E(F) com a chave K

Solução Proposta

Problemas do PDPr com criptografia homomórfica

- A avaliação de um algoritmo de hash criptográfico tem um custo computacional alto
- Minificação do ativo:
  - Publicar o arquivo encriptado com cifra simétrica
  - Realizar o protocolo de PDPr somente com a chave simétrica
  - Reduz a quantidade de bits na computação
- Mesmo com a minificação do ativo, a avaliação com cifra homomórfica ainda seria muito lenta para o caso de uso considerado. [3]

Solução Proposta

PDPr com hash homomórfico - aprimorado

- Sistema de encriptação GCrypt = (Enc, Dec, Gen)
- Similar ao OTP

$$Enc(m, k) = m + k$$

- Estende o OTP para um espaço no qual as entradas são $\mathbb{Z}$ módulo p

Solução Proposta

PDPr com hash homomórfico - aprimorado

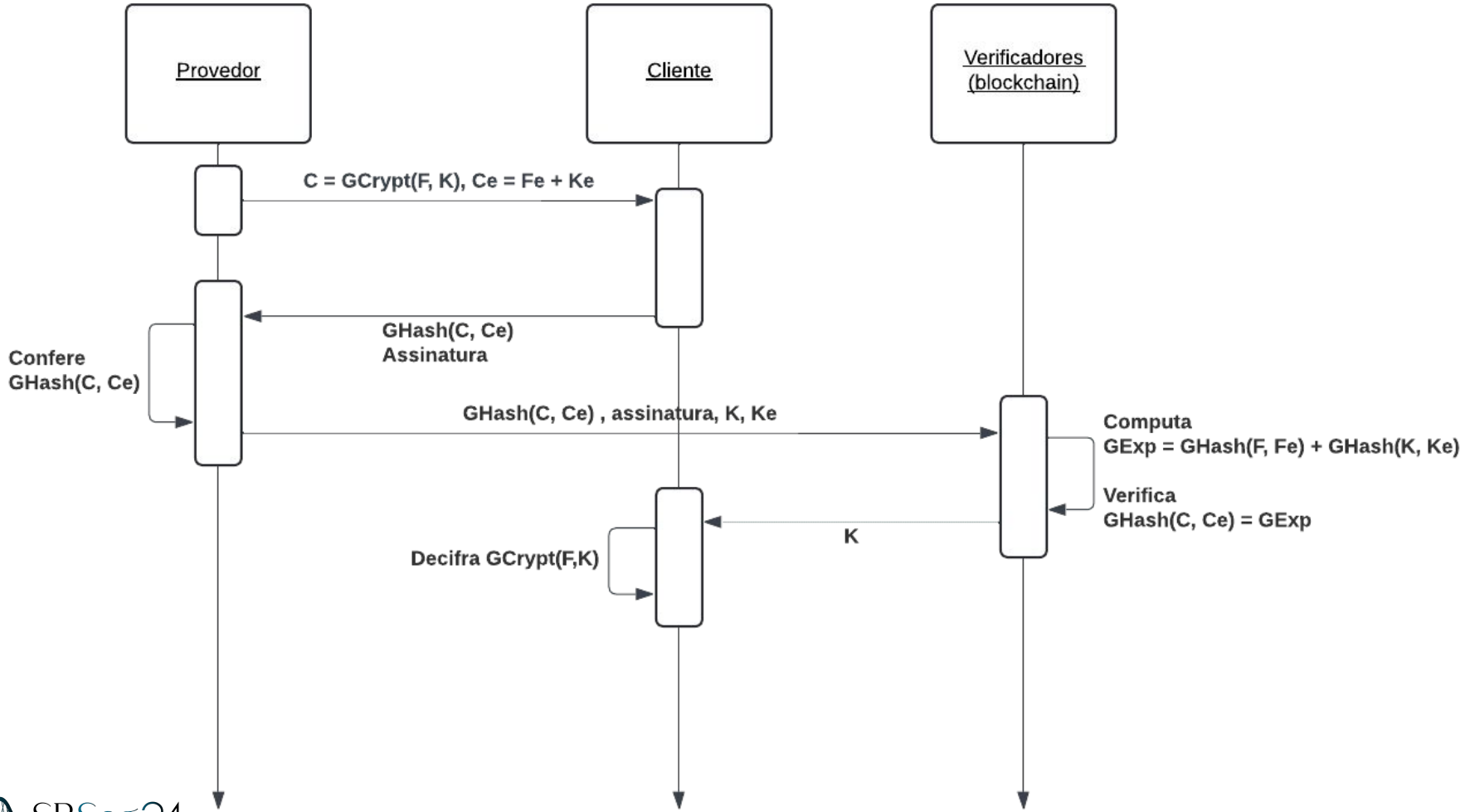
- Algoritmo de hash GHash

$$GHash(m, m_\epsilon) = m_\epsilon + \sum_{i=1}^{l} m_i \times LtHash(i)$$

- Propriedade de homomorfismo

$$GHash(a + b, a_\epsilon + b_\epsilon) = GHash(a, a_\epsilon) + GHash(b, b_\epsilon)$$

$$GHash(c, c_\epsilon) = GHash(m, m_\epsilon) + GHash(k, k_\epsilon)$$


Provedor
Cliente
Verificadores
(blockchain)
C = GCrypt(F, K), Ce = Fe + Ke
GHash(C, Ce)
Assinatura
Confere
GHash(C, Ce)
GHash(C, Ce) , assinatura, K, Ke
Computa
GExp = GHash(F, Fe) + GHash(K, Ke)
Verifica
GHash(C, Ce) = GExp
K
Decifra GCrypt(F,K)

Solução Proposta

PDPr com hash homomórfico - aprimorado

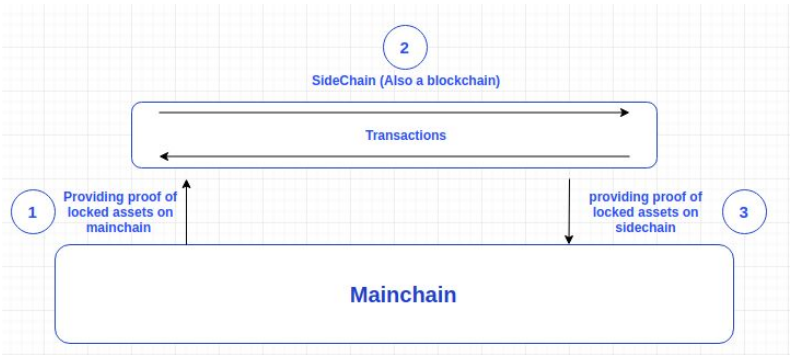
- O cliente não pode recuperar o conteúdo original a partir do texto cifrado, pois equivaleria a quebrar um esquema com segredo perfeito
- Os verificadores podem verificar que uma mensagem foi cifrada com uma chave, mas não podem recuperar a mensagem, pois equivaleria a quebrar uma instância de LWE

Solução Proposta

Design da blockchain

- Evitar criar uma nova moeda
- Implementar somente a verificação de downloads em uma *sidechain* de uma blockchain que seja popular

Cadeia de *download*
X
Cadeia de pagamento



Solução Proposta

Design da blockchain

- Mecanismo de popularidade
  - Contagem de *downloads* confirmados para um provedor na cadeia de *downloads*
- Proteções para o cliente no modelo de ameaças

Solução Proposta

Design da blockchain

- Mecanismo de consenso com *threshold Proof-of-Work* (PoW)
- Similar ao consenso de Nakamoto
- Várias provas de diferentes mineradores formam um comitê
- Transações incluídas na maioria das provas são inclusas no próximo bloco
- Forks: maior agregação de popularidade

Solução Proposta

Esquemas de download

- Definem quando e como os processos de download e pagamento serão embutidos nas transações da cadeia de *downloads*.
- Propostas de esquemas:
  - Pay-then-Download
  - Lock-then-prove
  - Lock-then-prove com validação externa

Avaliação - benchmarks

Intel i7 7th Gen - 2.9GHz - 8GB RAM

Tabela 1. Resultados do *benchmark* do GHash

Tamanho de entrada (blocos)	Tempo (ms)
128	15.55
256	30.63

Tabela 2. Resultados de *benchmark* do GCrypt

Tamanho de entrada (bits)	Tempo de cifração (ms)	Tempo de decifração (ms)
128	1.130	0.580
256	2.280	1.152

Avaliação - benchmarks

Intel i7 7th Gen - 2.9GHz - 8GB RAM

Tabela 3. *Benchmarks* do protocolo de PDPr

Operação	Tamanho de chave	Tempo (ms)
Geração de prova	128/256	17.59/34.94
Verificação de prova	128/256	289.5/576.5

Considerações finais

- Benchmarks dos algoritmos apontam para viabilidade da implementação
- Implementação da blockchain em andamento

Trabalhos futuros

- Melhorias no modelo de ameaças e formalização de provas de segurança
- Finalização da implementação e adição de mais resultados
- Aprimoramentos de performance e custo de *smart contracts*

Obrigado!

- Contato:

  jt.mat@hotmail.com

- Dúvidas e sugestões?

Referências

[1] Costa, F. Z. D. N., De Queiroz, R. J., Bittencourt, G. P., and Teixeira, L. (2022). Distributed repository for software packages using blockchain. IEEE Access, 10:112502112514.

[2] Bellare, M. and Micciancio, D. (1997). A new paradigm for collision-free hashing: Incrementality at reduced cost. In International Conference on the Theory and Applications of Cryptographic Techniques, pages 163–192. Springer.

[3] endoukha, A. A., Stan, O., Sirdey, R., Quero, N., and Freitas, L. (2022). Practical homomorphic evaluation of block-cipher-based hash functions with applications. In International Symposium on Foundations and Practice of Security, pages 88–103. Springer.

Patrocinadores do SBSeg 2024!

nic.br

cgi.br

Google

Tempest

CAPES

SiDi

FAPESP

zscaler™

BugHunt

CNPq

C.E.S.A.R

FACULDADE IBPTECH