Relazione Borsa di Ricerca

Enrico Pezzano

Ottobre 2025

1 Introduzione

La borsa di ricerca è stata dedicata allo sviluppo di *MedChain*, un prototipo di blockchain riscrivibile o oscurabile, impiegato per esplorare come i chameleon hash e le zero-knowledge proof possano supportare la gestione di dati sensibili. Ho scelto un ipotetico caso d'uso sanitario con cui fare leva sui requisiti di privacy e controllo. Il lavoro, svolto presso l'Università degli Studi di Genova, ha prodotto un simulatore Python, circuiti `circom`, contratti Solidity, tooling per IPFS e documentazione che descrivono come la piattaforma potrebbe operare in contesti regolamentati. Inoltre, il progetto di borsa implementa per esteso (non simulate) le proposte di Avitabile et al., basate su smart contract, integrando prove Groth16 per garantire la correttezza delle operazioni di censura.

2 Background

La ricerca ha avuto inizio da un modello di blockchain riscrivibile o oscurabile proposto da Ateniese et al., che sfrutta i chameleon hash per consentire modifiche controllate ai blocchi. La proposta di base di Ateniese è stata rielaborata dai lavori di Avitabile et al., che propongono una governance permissioned, basata su smart contract e su approvazioni da multipli ruoli. In letteratura il diritto all'oblio del GDPR viene spesso indicato come motivazione per studiare la censura selettiva di registri distribuiti; ho tratto ispirazione da questa normativa come scenario di riferimento. Il panorama di soluzioni esistenti spazia da prototipi accademici che simulano prove crittografiche a sistemi aziendali che rinunciano alla transparenza della blockchain. MedChain si colloca in mezzo.

3 Metodologia

3.1 Fasi di progetto

Il progetto è stato suddiviso in quattro fasi, dal benchmark di Ateniese fino all'integrazione in blockchain.

Step 0: Benchmark di Ateniese Ho iniziato implementando il modello di blockchain riscrivibile proposto da Ateniese et al., che costituisce la base teoretica del sistema. Per questa prima fase ho dovuto implementare un'implementazione Python completa dei chameleon hash, che consentono di censurare i blocchi, basandosi su trapdoor crittografiche. Ho inoltre implementato le strutture dati di base per blocchi e transazioni, riproducendo l'impalcatura essenziale di una blockchain. Per testare il sistema ho implementato un simulatore di consenso che replica meccanismi simili a Bitcoin, potendo così simulare la propagazione e la conferma di blocchi. Infine ho raccolto metriche di base che sono servite da riferimento per quantificare i miglioramenti introdotti dalle successive estensioni. Questo step ha fornito la baseline tecnica su cui poter confrontare quantitativamente gli avanzamenti del modello di Avitabile rispetto all'approccio originale.

Step 1: Simulazione Python della governance di Avitabile Partendo dal benchmark di Ateniese, ho esteso il sistema con una simulazione del modello di governance basato sugli smart-contract proposto da Avitabile et al. Lo scopo di questa fase era di poter validare l'intera architettura prima di affrontare l'implementazione crittografica reale, pertanto tutti i componenti crittografici sono stati inizialmente simulati. Ho quindi implementato uno strato di smart contract simulati mediante classi Python che emulano fedelmente il comportamento dei contratti, mantenendo lo stato di policy e approvazioni in strutture dati in memoria. Il sistema implementa le tre tipologie di richieste di censura previste dal modello (DELETE, ANONYMIZE, MODIFY), ognuna con soglie di approvazione differenziate: due approvazioni per le cancellazioni definitive, tre per le anonimizzazioni destinate alla ricerca, e una sola per le modifiche correttive. Per prendere le decisioni ho modellato un flusso di approvazione da più parti che coinvolge diversi ruoli (medico, amministratore, regolatore) con controllo degli accessi rigidamente basato su policy. Le prove zero-knowledge erano rappresentate da oggetti `SimulatedProof` che tornavano sempre esito positivo, potendo così eseguire i flussi logici senza affrontare il dispendio computativo delle prove. Ho inoltre implementato i flussi richiesti per la conformità al GDPR, in particolare per l'articolo 17 sul diritto all'oblio, ed ho integrato IPFS per la conservazione cifrata (AES-GCM) dei record censurati.

Lo step 1 ha consentito di validare in modo esaustivo l'architettura di governance, i workflow di approvazione e le interazioni da molteplici parti, con tutti i test end-to-end che sono stati eseguiti con successo in modalità simulata prima di proseguire verso la fase crittografica.

Step 2: Prove Zero-Knowledge Reali (Fase 1 / Bookmark1) La Fase 1, battezzata intenzionalmente nel codice come `Bookmark1`, ha sostituito a tappeto ogni componente simulato con prove crittografiche reali, rimuovendo così in modo integrale il codice di simulazione dai percorsi di produzione. Ho progettato e implementato un circuito `circom` completo per il sistema di prove Groth16, che espone 16 segnali pubblici che trasportano il nullifier, gli hash di consistenza e i dati di policy necessari alla verifica. Per mappare record sani-

tari sintetici in campi compatibili con la curva ellittica BN254 ho sviluppato il `MedicalDataCircuitMapper`, componente che garantisce la serializzazione deterministica e la corretta conversione dei dati sanitari in ingressi di circuito validi. Ho poi realizzato uno SNARK manager che agisce come wrapper Python di `snarkjs`, gestendo in modo trasparente per il resto del sistema l'intero processo di generazione di witness e prova. Parallelamente ho implementato un generatore di prove di consistenza che copre cinque tipi di verifica: la consistenza del Merkle-Tree, l'integrità della catena di hash, la validità dello stato del contratto, il corretto ordinamento delle transazioni, e l'integrità complessiva dei dati. Per assicurare l'affidabilità dell'implementazione ho sviluppato una suite di test che conta oltre 40 test unitari per il mapping del circuito e più di 5 test di sistema per la generazione delle prove.

Questa fase ha gettato le basi per la generazione di prove crittograficamente solide, raggiungendo tempi di generazione compresi tra i 5 e i 10 secondi per richiesta, e garantendo una verifica off-chain perfettamente funzionante prima di passare a verificare sulla blockchain.

Step 3: Verifica On-Chain (Fase 2 / Bookmark2) La Fase 2, intenzionalmente denominata nel codice come `Bookmark2`, ha portato le prove sulla blockchain, completando l'integrazione con gli smart contract e la verifica on-chain. Ho implementato tre contratti Solidity fondamentali: `NullifierRegistry` per impedire attacchi replay tramite il monitoraggio dei nullifier utilizzati, `MedicalDataManager` per l'enforcement delle policy e la verifica delle prove, e `RedactionVerifier` per la verifica crittografica delle prove Groth16 direttamente sulla blockchain. Per interfacciare il backend Python con gli smart contract ho realizzato un'integrazione in `medical/backends.py` che si occupa di inviare le prove mediante chiamate Web3, occupandosi della corretta serializzazione dei parametri e della gestione delle transazioni. Ho inoltre automatizzato l'intero processo di deployment con script Hardhat che consentono il rilascio su più reti, salvando automaticamente gli indirizzi dei contratti deployati per poterci interagire più agevolmente. Infine l'implementazione prevede pipeline di sottomissione end-to-end che coprono l'intero percorso dalla generazione locale della prova fino alla registrazione on-chain della richiesta di censura, con oltre 15 test d'interazione che verificano le interazioni con i contratti e i flussi.

Stato attuale Tutti i componenti software sono testati attraverso tutti e quattro gli step. L'implementazione dimostra una padronanza della tecnologia blockchain (Step 0), architettura di governance (Step 1), SNARK Groth16 e design di circuiti `circom` (Step 2), sviluppo di contratti Solidity e integrazione EVM (Step 3). Il lavoro che resta è di natura meccanica: la ricompilazione degli artefatti del circuito per attivare il percorso di verifica a 16 segnali descritto nel codice sorgente.

3.2 Architettura e stack tecnologico

L'architettura mantiene separati i domini tecnici al fine di favorire uno sviluppo modulare ed un test:

- **Simulatore Python**: coordina richieste di censura, approvazioni multi-parte e generazione delle prove; gestisce il workflow dalla creazione del record alla verifica finale
- **Circuiti circom**: verificano hash di policy, radici di Merkle e coerenza tra dati originali e redatti; il design attuale definisce 16 segnali pubblici per nullifier e dati di consistenza
- **Contratti Solidity**: validano le prove Groth16, registrano i nullifier per prevenire replay, e emettono eventi di audit immutabili sulla blockchain
- **IPFS**: gestito via `ipfshttpclient`, assicura la conservazione cifrata (AES-GCM) dei record censurati, separando storage pubblico da dati sensibili
- **Web3.py**: fornisce il collegamento con reti EVM quando il backend è configurato per l'esecuzione on-chain

La progressione Step 0 → Step 1 → Step 2 → Step 3 riflette l'evoluzione da baseline semplice a sistema completo: fondamenta blockchain, simulazione della governance, crittografia reale, integrazione on-chain. Ogni step mantiene retrocompatibilità con i precedenti, potendo così testare componenti isolati o il sistema completo.

3.3 Workflow

Una richiesta di censura attraversa il sistema con il seguente workflow completo:

1. **Creazione richiesta**: un attore autorizzato (medico, amministratore, regolatore) crea una richiesta specificando tipo (DELETE/ANONYMIZE/MODIFY), paziente target, motivazione e ruolo richiedente
2. **Validazione policy**: il sistema verifica che il ruolo sia autorizzato per il tipo di censura e che i dati esistano
3. **Generazione prova SNARK**:
    - Il circuit mapper serializza il record originale e redatto in formato JSON
    - I dati vengono convertiti in elementi di campo compatibili con BN254
    - `snarkjs` genera witness e prova Groth16 (5–10 secondi)
    - La prova viene verificata off-chain prima di procedere
4. **Generazione prova di consistenza**: il sistema calcola gli hash di stato pre/post censura, verifica che solo i campi autorizzati siano modificati, e genera commitment crittografici

5. **Raccolta approvazioni**: la richiesta entra in coda fino al raggiungimento della soglia configurata (DELETE: 2 approvazioni, ANONYMIZE: 3, MODIFY: 1)

6. **Sottomissione on-chain** (Step 3):

   - Parsing dei componenti Groth16 (a, b, c, publicSignals) per formato Solidity
   - Estrazione del nullifier dai segnali pubblici
   - Chiamata a `requestDataRedactionWithFullProofs` del contratto `MedicalDataManager`

7. **Verifica on-chain**:

   - Controllo che il nullifier non sia già stato usato (prevenzione replay)
   - Verifica crittografica della prova tramite contratto `RedactionVerifier`
   - Validazione degli hash di consistenza
   - Registrazione del nullifier per uso futuro

8. **Esecuzione e audit**: se tutte le verifiche hanno esito positivo la censura viene eseguita e il sistema emette eventi di audit che registrano timestamp, attori coinvolti, hash delle prove, e stato pre/post modifica

L'intero flusso è tracciato tramite eventi e log strutturati, potendo così ricostruire in modo esaustivo tutta la storia delle decisioni per audit esterni. I demo script illustrano scenari quali una cancellazione GDPR, un'anonimizzazione per la ricerca, e delle correzioni puntuali con approvazioni da più ruoli.

4 Risultati

Il deliverable consegnato fornisce un'infrastruttura completa sviluppata attraverso quattro step progressivi, accompagnata da documentazione esaustiva e da test automatizzati che ne attestano il funzionamento.

Durante lo Step 0 è stata completata l'implementazione della baseline di Ateniese, realizzando un simulatore Python completo del modello di blockchain redattabile con chameleon hash. Questa implementazione ha fornito le metriche di confronto necessarie e le fondamenta tecniche su cui costruire gli step successivi. Proseguendo con lo Step 1, ho sviluppato una simulazione completa della governance proposta da Avitabile, implementando workflow multi-parte con policy differenziate per tipo di censura e controllo degli accessi basato su ruoli, potendo così validare l'intera architettura prima di affrontare la complessità dell'implementazione crittografica reale.

Lo Step 2 ha segnato il passaggio alle prove crittografiche concrete: tutte le operazioni di redazione utilizzano ora prove Groth16 generate con `circom` e `snarkjs`, verificate off-chain con esito positivo. Il circuito è stato progettato

per esporre 16 segnali pubblici che trasportano il nullifier e i dati di consistenza necessari alle verifiche. Con lo Step 3 ho portato la governance sulla blockchain, implementando contratti Solidity che codificano soglie di approvazione differenziate per ciascun tipo di operazione: le cancellazioni definitive richiedono due approvazioni, le anonimizzazioni per la ricerca ne richiedono tre, mentre le modifiche correttive necessitano di una sola approvazione. Questi contratti tengono traccia di tutte le decisioni attraverso eventi immutabili che facilitano l'audit.

Per garantire la sicurezza del sistema, il contratto `NullifierRegistry` previene gli attacchi replay impedendo il riutilizzo di prove già accettate, registrando ogni nullifier insieme al timestamp e all'indirizzo del sottomittente.

L'intero processo di sviluppo e deployment è stato automatizzato attraverso script Hardhat, task Makefile e workflow di test che permettono di ricostruire facilmente gli ambienti di sviluppo. La qualità del codice è stata validata con oltre 40 test unitari e più di 15 test d'integrazione, con report di coverage completi sia per Python che per Solidity.

L'integrazione con IPFS garantisce lo storage cifrato dei dati tramite cifratura AES-GCM dei record censurati, mantenendo una netta separazione tra dati pubblici conservati su IPFS e informazioni sensibili registrate on-chain, con mapping crittograficamente verificabili tra i due livelli. Per dimostrare il funzionamento completo del sistema ho sviluppato tre script dimostrativi che illustrano workflow end-to-end: una pipeline IPFS con dati censurati, un processo di censura con approvazioni multi-parte, e la validazione di consistenza delle operazioni.

Le performance del sistema sono state stimare tramite misurazioni preliminari: la generazione delle prove richiede tra 5 e 10 secondi, la verifica on-chain di una prova Groth16 consuma circa 250k gas, mentre una richiesta completa necessita di circa 350k gas. In un contesto permissioned la latenza end-to-end si mantiene sotto i 15 secondi, rendendo il sistema utilizzabile per casi d'uso reali.

Gap tecnico ancora aperto Rimane ancora un importante vincolo tecnico: gli artefatti di circuito presenti in `circuits/build/` (WASM, R1CS, zkey) espongono ancora un solo public signal, residuo di configurazioni precedenti agli step 2 e 3. Il codice sorgente del circuito definisce correttamente 16 segnali pubblici, e tutto il codice Python è già configurato per gestirli, ma finché i circuiti non vengono ricompilati con `circom` v2 e i contratti aggiornati per accettare vettori `uint[16]`, le prove non possono essere completamente convalidate on-chain.

Questo è un gap meccanico, non architetturale: tutta l'infrastruttura di software è pronta, i circuiti sono scritti, i contratti implementati, i test completati. Il problema è che il compiler `circom` non è installato nell'ambiente di sviluppo, e pertanto non si possono rigenerare gli artefatti. Nella documentazione (Appendice A.1.3 del deliverable esteso) c'è una pratica guida in 10 passi per portare a termine la ricompilazione.

Pur rimanendo bloccati sulla verifica on-chain end-to-end l'infrastruttura è completamente funzionate per la generazione e la verifica off-chain delle prove Groth16, il testing di tutti i workflow di governance e approvazioni, la sim-

ulazione completa del modello di Avitabile, lo sviluppo e il test dei contratti Solidity, e l'adattamento del sistema ad altri domini con dati sensibili.

5 Conclusioni e futuri sviluppi

Il lavoro svolto con MedChain nei quattro step di sviluppo mi ha permesso di consolidare competenze su blockchain permissioned, crittografia applicata con SNARK Groth16 e chameleon hash, architettura di sistemi distribuiti, e sviluppo di smart contract. La progressione metodica dal benchmark di Ateniese alla simulazione della governance, sino all'implementazione di prove crittografiche reali e all'integrazione on-chain, ha rappresentato un rigoroso approccio che privilegia la validazione dell'architettura tramite simulazioni prima di affrontare la complessità dei componenti crittografici.

Il caso d'uso sanitario ha fornito un utile banco di prova per esplorare requisiti di privacy, conformità GDPR e controllo degli accessi multi-ruolo, pur mantenendo un carattere ipotetico e appropriato a scopo di ricerca.

Il lavoro ha prodotto oltre 60 pagine di documentazione tecnica, tre script dimostrativi completi, una suite di oltre 55 test automatizzati, ed anche degli strumenti che consentono di agevolare l'onboarding di nuovi contributori al progetto.

Per completare l'implementazione end-to-end manca la ricompilazione dei circuiti con `circom` v2 per generare artefatti a 16 segnali pubblici, l'aggiornamento dell'interfaccia del contratto `RedactionVerifier`, e l'esecuzione dei test completi su blockchain. Nella documentazione c'è una pratica guida per questi passaggi, che penso poter affrontare in 2–4 ore di lavoro, e successivamente sarà possibile raccogliere metriche sperimentali di performance su reti di test con carico realistico.

Futuri sviluppi di medio termine potrebbero essere l'implementazione di batch verification per ridurre i costi di gas, l'ampliamento degli scenari di test tramite edge case e attacchi simulati, l'aggiunta di strumenti di monitoring e observability, ed una possibile interfaccia web per la governance. L'ottimizzazione delle performance tramite caching degli artefatti di `circom` e parallelizzazione della generazione delle prove sarebbe un ulteriore importante miglioramento.

A lungo termine il framework MedChain può essere adattato a domini diversi da quello sanitario come registri catastali con correzioni notarili, supply chain con gestione di recall, sistemi di identità digitale con aggiornamenti privacy-preserving, ed archivi legali con redazioni autorizzate. La separazione modulare tra layer di governance, crittografia e blockchain rende questo adattamento possibile mantenendo le garanzie crittografiche di base. L'adozione di tecnologie emergenti come recursive SNARKs, threshold cryptography e trusted execution environments potrebbe estendere le capacità del sistema, fornendo un solido trampolino di lancio per ricerca futura su blockchain redattabili con governance complessa e conformità normativa.

Data:

November 4, 2025

Firma Studente

_____

Firma Responsabile

_____