Informazioni utili per il progetto

☑ Reviewed	☐

File e informazioni - Prelazione

Qui possiamo trovare alcune delle informazioni necessarie per comrpendere il key trasparency e i vari standard su cui si basa:

- [ ] Vedere key trasparency in se: presentazione ufficiale PEPR e presentazione ufficlae RWCc on pdf;
- [ ] Come è stato sviluppato : meta e paper overview;
- [ ] Cos'è crpto end to end : link;
- [ ] Repository ufficiale meta : github;
- [ ] Presentazioni : RWC and PEPR;
- [ ] RUST perchè codice in rust quindi continuare lo studio: StudiaRust

Scaletta

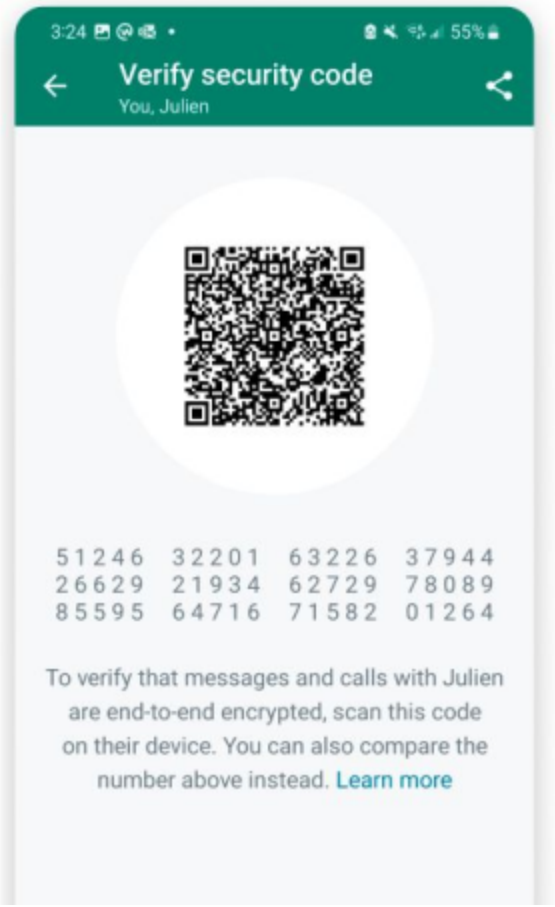
- Overview sul protocollo di key exchange di whatsapp e approfondimento sui vari protocolli utilizzati (Seemless,parakeet,coniks),
- Approfondimento sulla libreria AKD che fornisce un'implementazione di una directory di
  chiavi controllabile dando anche una API utilizzabile per dei test.
    - con approfondimento della libreria intendo anche una parte sperimentale in cui la utilizzo . Potrei anche inserire una parte puramente informativa, ricavata da una pagina web che discuteva di una vulnerabilità nella crittografia di whatsapp , riguardanti nello specifico ad alcuni metadati visibili e come alcuni governi potrebbero sfruttarla

Whatsapp key trasparency

Cerchiamo di capire ic oncetti ad alto livello.

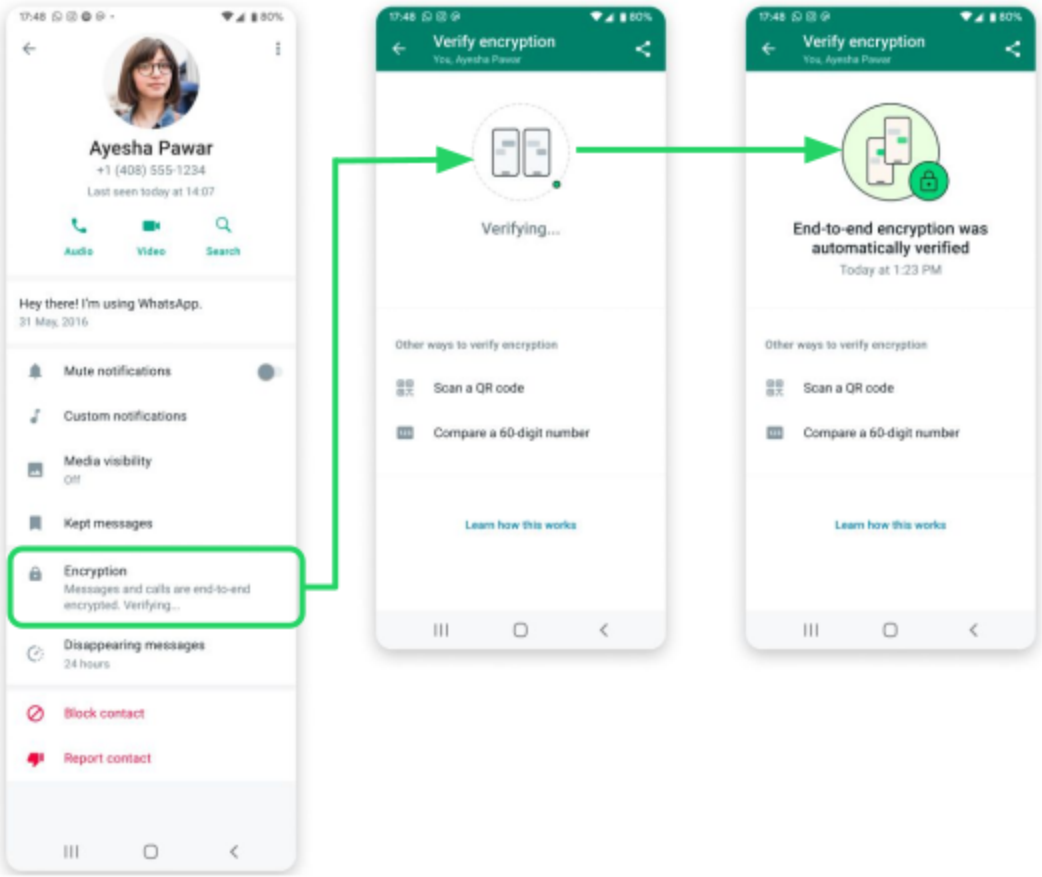
Phone #s	Public Keys
Alice	pk_Alice
Bob	pk_Bob
…	…

Andiamo a considerare whatsapp come un sistema crittografato che funziona con due perosne che comunicano con l'uso di chiavi pubbliche , possono cosi dopo aver ottenuto le chaivi pubbliche eseguire un protocollo di scambio di chiavi per stabilire una sessione sicura tra loro in modo che nessuno tranne i punti finali della sessione possono decifrare i messaggi. Questo vuol dire che il server tra i due utenti manterrà una mappatura sugli utenti , ad esempio i numeri di telefono nel caso di whatsapp le chiavi pubbliche e le distribuirà su richiesta . Ad esempio alice vuole contattare BOB for the first time , inserirà il numero di BOB in whtaasapp e il server darà la publick key. Ora , questo punto davvero importante per la sessione è che dia la chiave corretta , corrispondente a bob il server di whatsapp. Quindi ad esempio se dovesse dare la pub key corrispondente a qualche altra private key , consentirebbe al sever di generare un MITM sulla sessione tra Alice e bob.



Il modo in cui Alice e Bob , ma comunque il modo in cui la maggior parte delle app di messagistica risolve questo problema di chiavi correttamente erogate è esponendo una sorta di verifica manuale della pub key tra utenti . Il modo in cui funziona è sostanzialemnte di incontrarti di persona o tramite una chiamata e scan il qr code del contatto . Il qr è costruito fondamentalmente come un hash della chiavi pubbliche per entrambi gli user. Quindi con questo approccio attuale ci sono diversi problemi , uno dei quali che questi codici cambaino al cambiamento del dispositivo , oppure se tu aggiungi un altro dispositivo al tuo account e anche se immagini di avere una chat di gruppo di perosne con ad esempio 10 persone , per avere la garanzia delle ciavi devi scan i qr di tutti i membri e non solo di una. Quindi molte volte le persone cambiano spesso dispositivo e tutto questo diventa difficile. Non sai nemmeno se sai che lo faranno di persona. Sopratutto in sistuaizoni in cui non sarebbe possibile intervenire di persona quindi come un informatore che vuole contattare un giornalista , in un paese contrllato dal governo . Quindi non avrebbe la certezza di star parladno con il gionralista . Al momento

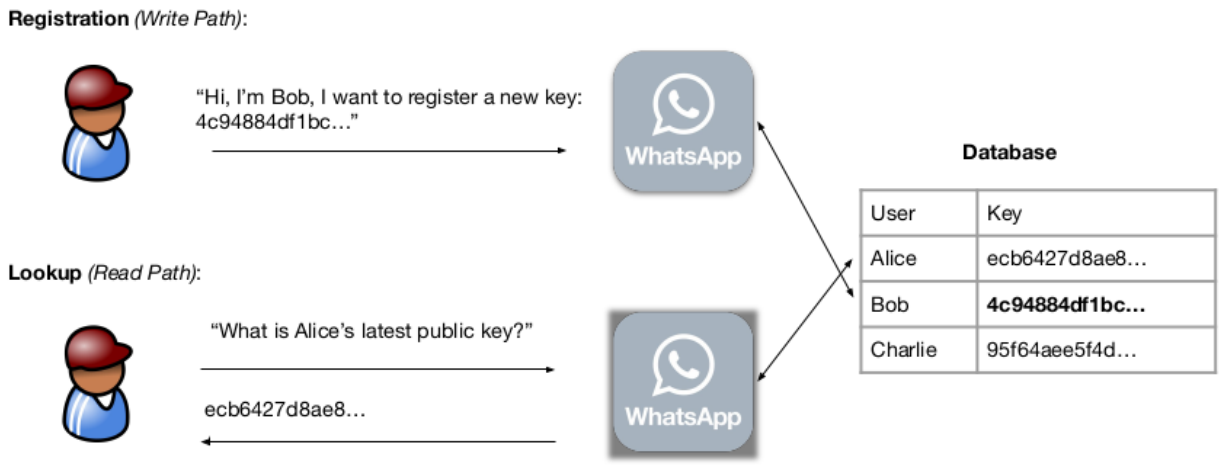
nella app mesh crittografate , potrebbe esserci un agente sottocopertura che convince whatsapp a darti la pubkey sbagliata e ciolpirti , ma è qui che entra in gioco la trasparenza della chiave.



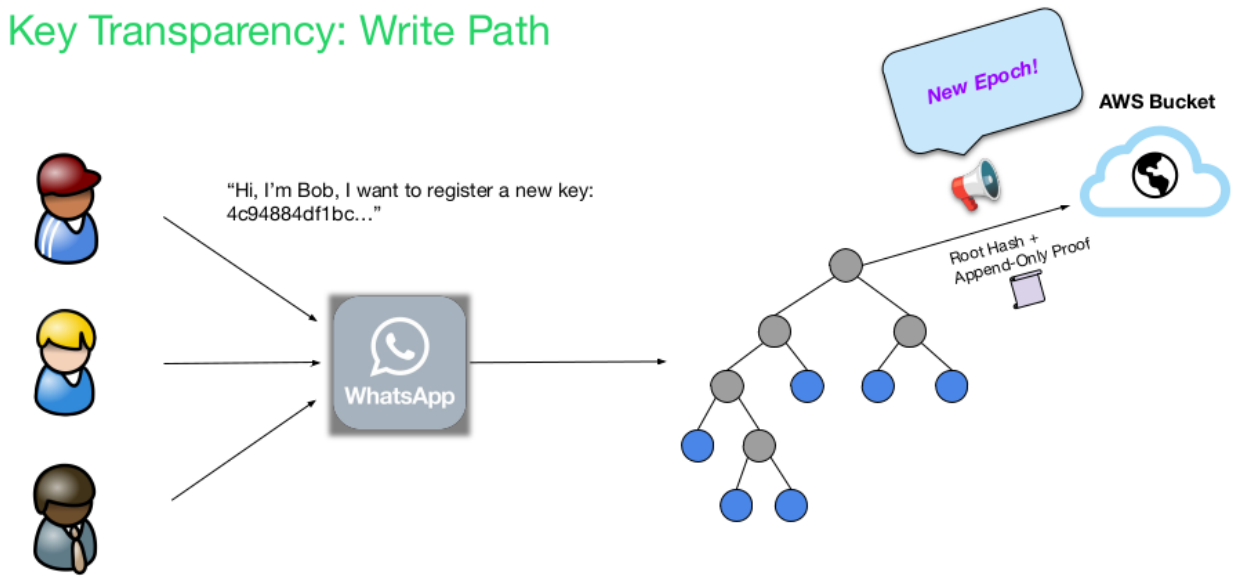
Ecco che entra quindi in gioco la key trasparency . Questa è una soluzione per risolvere questo problema e l'idea di base è che , vogliamo fonrire un mezzo per eseguire le autenticazione della chiave senza che i due utenti si incontrino , quindi in modo piu automatico .
Il server renderà pubblica un DB di public key e quindi consentire agli utenti di verificare crittograficamente che il server sti adistirbuendo queste pub key in modo coerente. Ad esempio il server può impeganrsi in questo dDB di numero di telefono di chaivi pub ad un certo punto alice può controlalre la sua pub key nel db rispetto . Questo rende automatico il processo di verifica perce anceh se facciamo affidamento al server come intermediario per il passaggio di un messaggio , ora abbiamo questo meccanismo crittografico per impedire al server di fornire chiavi incoerenti ,ovvero il controlo da parte dell’users.

Infrastructur

Ecco come sarebbe l'infrastruttura normale di registrazione e ricerca (lookup)

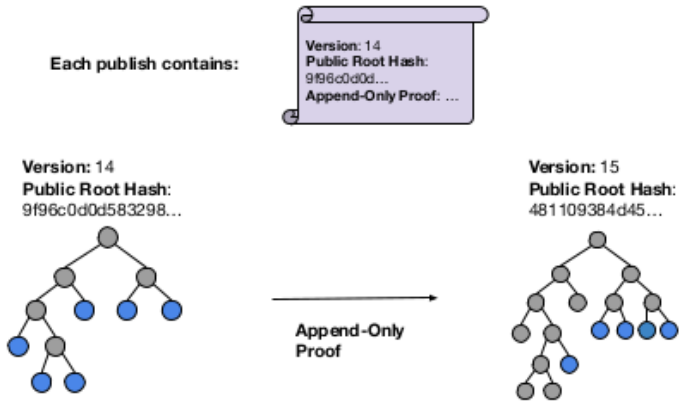


Ecco come invece rsulta con il key trasparency :



OK quindi , lavorando a livello alto come funziona ? Vediamolo.
Come otteniamo queste garanzie. L'idea è che per la registrazione il server mantiene una sorta di merkle tree, dove le foglie di ogni albero rappresentano le verisoni di ogni aggironamento chaive ottenuto . Immaginiaamo di avere un miliardo di utenti che hannoa ggiornato la chaive tutti per tre volte, avremo circa

3 miliardi di foglie nel markle tree e poi l'albero sarà orgnaizzato in modo da continuare ad esguire l'hashing de due nodi vicini finche non avremo un singolo hsah root in cima all'albero che rappresenta un commit per l'intero database di tutti gli aggiornamenti per tutte el chiavi e poi abbiamo questa nozione "epoch " per la pubblicazione in cui aspetteremo e raccogliamo tutti gli aggiornamenti avvenuti negli ultimi minuti e ad esempio annunceremo una nuova epoch su qualche endopoitn pubblico che idealmente chiunque può leggere.



Append-only proofs guarantee that we manage the database consistently

Quindi ,queste ricevute epoch conterranno tre cose . Ci sarà un nuovo numero di versione associato all'epoca ,c'è un nuovo hash root pubblico dell'ultimo markle tree e poi c'è una chiamata prova di audit o prova di sola aggiunta che è una prova che l'albero di merkle in questa epoca attuale è stato derivato in modo coerente dall'albero di merkle nell'epoca precednte.

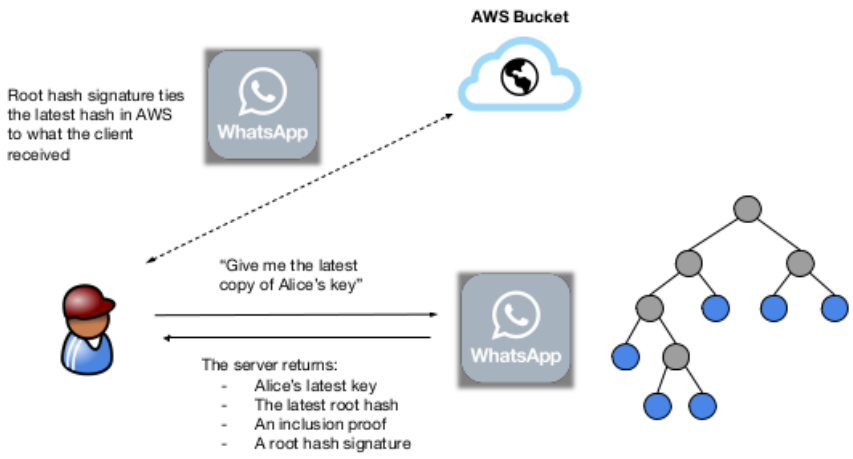
L'audit in Key Transparency implica diversi processi chiave:

1. **Verifica delle Chiavi Pubbliche**: Gli utenti e i server possono verificare che una chiave pubblica appartenga effettivamente a un determinato utente controllando il registro di Key Transparency. Questo garantisce che le chiavi non siano state alterate o sostituite.
2. **Consistenza del Registro**: Il registro di Key Transparency è progettato per essere append-only, il che significa che una volta aggiunti, i record non possono essere modificati o eliminati. Gli audit possono verificare che il registro mantenga questa proprietà di consistenza.

3. **Prove di Inclusione**: Gli utenti possono richiedere prove che le loro chiavi pubbliche siano effettivamente incluse nel registro. Questo può essere fatto attraverso tecniche crittografiche che dimostrano che una chiave specifica è parte del registro senza rivelare l'intero contenuto del registro.
4. **Verifica della Trasparenza**: I client di WhatsApp possono eseguire controlli regolari per verificare che le chiavi pubbliche degli utenti con cui comunicano non siano state sostituite o alterate. Questo può includere confronti di hash o altre prove crittografiche.

   L'audit di Key Transparency è fondamentale per mantenere la fiducia nella sicurezza delle comunicazioni. Garantisce che le chiavi di crittografia non siano state compromesse e che la privacy e la sicurezza delle comunicazioni rimangano intatte. Questo è particolarmente importante in un'applicazione come WhatsApp, dove la riservatezza delle comunicazioni è una priorità.

Senza aver cancellato la cronologia che è stata registrata o senza modifica di alcun valore. Queste entrate di epoca (ricevute) sono pubblicate in modo che chiunque possa verificare che stiamo gestendo queste chiavi onestamente e questo è quello veramente importante , aquesto proposito che sia importante che le ricevute non contengano nessuna informazione riservata e sensibile , come numero di telefono o anche le stesse ciavi pubbliche.



Ora per gestire l'aspetto delle richieste utilizzando la trasparenza delle chiavi Bob prima raggiunge l'endpoint publico per otteneere l'ultima epoca pubblicata e

anche questo hash root associato all'epoca. Bob chiedera a whatsapp di ottenere l'ultima copia della chiave di alice che corrisponde a questo hash di root e i server whatsapp possono quindi usare la rappresentazione dell'albero di merkle per restituire la chiave appropirialta di Alice , ma anche la ciave appropriata per Alice , anche una prova di inclusione che bob può verificare e che gli affermerà che la chiave di alice è inclusa nell'albero di merkle il cui hash corripsonde a ciò che ah chiesto .
Qunidi .

Quindi otteniamo il root hash che, ricordiamo cambia in base ad ogni epoca che ogni epoca viene inseirita ogni 5 minuti, per sapere se le chiavi sono aggiorante o meno . Si ricava da questo server aws bucket dove sono hostate tutti i root hash di ogni epoca. Ricavato il root hash il client di whatsapp tramite questo root ottiene dai server whatsapp la chiave pubblica di alice che usera per inviare i messaggi ad alice cifrati . Insieme alla chiave viene inviata anche una prova , in modo che il client possa usare quest aprova (tipicamente degli hash intermedi) che possono essere usati per ricavare il root hash e confrontarlo con quello che abbiamo ottenuto in modo da avere la certezza della correttezza delle chiavi diffuse.

Utilizzando la Merkle proof fornita dal server di WhatsApp insieme alla chiave pubblica, puoi risalire dall'hash della chiave pubblica fino alla radice del Merkle Tree (root hash). Poi puoi confrontare questa radice calcolata con il root hash ottenuto dall'AWS S3 bucket per verificare l'integrità e la correttezza della chiave pubblica. Ecco un riepilogo dettagliato del processo:

Passaggi per Verificare la Chiave Pubblica

1. **Richiesta della Chiave Pubblica**:
    - Bob invia una richiesta al server di WhatsApp per ottenere la chiave pubblica di Alice.
2. **Risposta del Server**:
    - Il server di WhatsApp risponde con:
        - La chiave pubblica di Alice (`public_key_Alice`)
        - Una Merkle proof (`merkle_proof_Alice`), che include gli hash intermedi necessari per risalire dalla chiave pubblica alla radice del Merkle Tree.

3. **Recupero del Root Hash**:
    - Bob recupera il root hash più recente da un AWS S3 bucket pubblico:

      ```
      GET <https://s3.amazonaws.com/my-key-transparency-bucket/root_hash_epoch_123.txt>
      ```

    - Questo root hash è chiamato `root_hash`.
4. **Verifica della Chiave Pubblica con la Merkle Proof**:
    - Utilizzando la chiave pubblica di Alice e la Merkle proof, Bob ricostruisce il percorso nel Merkle Tree fino alla radice.
    - La Merkle proof fornisce una serie di hash intermedi che, combinati con l'hash della chiave pubblica di Alice, permettono di risalire fino alla radice.

Esempio di Calcolo del Root Hash

Supponiamo che il Merkle Tree sia organizzato come segue:

```
        root_hash
         /      \\
    hash_AB   hash_CD
     /   \\     /   \\
hash_A hash_B hash_C hash_D
```

- `hash_A` è l'hash della chiave pubblica di Alice.
- `hash_B`, `hash_C` e `hash_D` sono gli hash delle chiavi pubbliche di altri utenti.

Bob riceve:

- `public_key_Alice` con `hash_A`
- `merkle_proof_Alice` contenente `hash_B` e `hash_CD`

Bob verifica:

1. Calcola `hash_AB` combinando `hash_A` e `hash_B`:

   ```
   hash_AB = hash(hash_A || hash_B)
   ```

2. Calcola `root_hash` combinando `hash_AB` e `hash_CD`:

```
calculated_root_hash = hash(hash_AB || hash_CD)
```

3. Confronta `calculated_root_hash` con `root_hash` recuperato dall'AWS S3 bucket:

```
if calculated_root_hash == root_hash:
    # La chiave pubblica di Alice è verificata come corret
ta e inclusa nel Merkle Tree
else:
    # La chiave pubblica di Alice non è verificata, potreb
be essere stata alterata
```

Vantaggi della Verifica

- **Sicurezza**: Garantisce che la chiave pubblica non sia stata manomessa, confermando che è inclusa nel Merkle Tree registrato.
- **Trasparenza**: Gli utenti possono verificare autonomamente l'integrità delle chiavi pubbliche utilizzando le Merkle proof.
- **Efficienza**: Le Merkle proof richiedono poche informazioni aggiuntive e sono efficienti dal punto di vista computazionale.

Conclusione

Il processo descritto garantisce che, utilizzando la Merkle proof fornita insieme alla chiave pubblica, puoi risalire fino alla radice del Merkle Tree e confrontare questa radice con il root hash ottenuto dall'AWS S3 bucket. Se i due hash coincidono, puoi essere sicuro che la chiave pubblica è autentica e corretta, fornendo un alto livello di sicurezza per le comunicazioni su WhatsApp.

Ora parliamo di alcune sfide che si affrontano durante l'implementazione pratic adel key trasparency

Principali sfide

MAggiori sfide da affrontare durante l'implementazione delle chiavi di trasparenza nella pratica. uindi una cosa è la distribuzione coerente di questi hash root. Quinid in sostanza tutta questa verifica crittografica eseguita dagli utenti sarà sempre cotraria a un impegno pubblico . Qunidi l'albero di merjkle che puoi vedere come una riduzionedel problema di servire in modo coerente come un database in continua manutenzione per servire cosantemente un singolo hash da 32 byte che continuaiamo ad aggiornaarre ogni cinque minuti . ma abbiamo ancora bisogno di un modo per distribuire questi hash in modo coerente e idealmente quello che faremmo è su un asorta di quorum fidato distribuito simile ad una blockchain che supporta in modo semplice per i clienti di verificare che le cose che pubblichaimo su diessa siano valide. e non lo facciamo , per ora si fa un servizio interno che vale una chiave di firma in un contatore e ogni volta che c'è una nuova epoch pubblicata eseguiamo il ping di questo servizio con un nuvo hash root e fa saltare questo contatore interno e poi ci da una firma con la garanzia che ci danno solo una firma per epoca e noi serviamo la firma ai client, quindi in pratica se puoi fidarti che stiamo eseguendo il servizio in questo modo allora ogni firma attesta il fatto che c'è solo un hash root firmato per ogni epoca che essenzialmente attenuerebbe l'equivoco del server ma in modo critico , è necessario che tu abbia fiducia nel fatto che lo stiamo eseguendo inetrneamente , quidni ci piacerebbe avere un modo verificabile esternamente per eseguire questo sevzio.

Un'altra sfida da affrontare è quindi in questo momento come abbiamo descritto la trasparenza delle chiavi , abbiamo la garanzia che se alice e bob sono online nello stesso momento e verificano le rispettive chiavi nella stessa epoca allora abbiamo l agaraznia di corenza. MA in pratica è irragionevole pensare che due utenti confronterranno l'epoca nel momento in cui sono online nelllo stesso momento controlalndo constantemente le chiavi degli altri e quindi quello che vogliamo è una garanzia che funzioni con le eppoch e un modo in cui gestitscono questo , che è stato suggerito nella letteratura accademica , è consentire un cosiddetto " controllo della cronologia delle chiavi ". L'idea è che gli utenti possono controllare la cronologia peridoidcamente delle proprie chiavi e quindi eseguire controllli di ricerca sulle chiavi dei contatti in moodo corretto , in questo modmo sto controlladno che whatsapp abbia rappresentato sempre correttaemente la mia chiave e non hanno fatto qualcosa priam vlaida poi no poi si .
Poi se tutti fanno check della prorpia cornologia , qunidi utti possiamo controllare

l'ultima versione delle chiavi dei nostri contatti e quidni abbiamo le garanzie ideali .
Il problema nel key trasparency è che nel nostor appuntamento attuale non l'abbiamo fatto , trovare un modo per far emegere la cronologi adelel modifiche chiave per gli utenti per molteplici ragini . Per due motivi : uno non è un qualcosa che ci aspettiamo necessariamente che le persone ricordino l'ultima volta che hanno cambiato i loro dispositivi o sinao in grado di usare tali informazioni .

Ci sono alune idee su come supportare la storia ma sono qualcosa su si sta lavorando .

Ad oggi ci sono 50k di epoche pubblicate ,,20 miliardi di nodi di merkle che mantengono lato server. LA frequenza ddi èubblicazione è ogni 5 minuti e si aggiungono 100k nuove voci all'albero . Una cosa interessante è che la nostra operazione di pubblicaizone ha l'obiettivo di ogni 5 minuti che è il tempo necessario per indicizzare questo database deve essere inferiore ai 5 minuti e molto meno di 5 minuti e in praticità questi sistemi molte volte immagino che altre implementazioni ho sentito problemi con la capacita di gestire questi alberi e il loro stato è un qualcosa su cui si concentrno maggiormente . E questo ha guidato nella scelt sa dell'algoritmo delel cose o il modo in cui si organizza l'albero epoi ic'è un un grande svantaggio nell'impleementeazione , che le prove di audit sono davvero grandi qunidi si pubblicano le ricevute di epoch ogni 5 minuti perchè l'audit dimostra essenzialmente che ci sono alcuni dettagli che non coprono in modo in cui sono costruitii ma in questomomento sono come centinaia di megabyte e non è roprio fattibile aspettarsi che tutti scarichino centinaia di megabyte per verificare che lo stiamo eseguendo correttamee , forse per alcuni lo farebbero ma si deve trovare un mood per ridurre.

Protocolli e Hash utilizzati nel Key Trasparency

Per comprendere al meglio il key trasparency dobbiamo chairire alcuni cocetti basilari

Albero di Merkle

I merkle tree sono una struttura dati creata con l'obiettivo di facilitare la verifica di grandi quantità di dati organizzati mettendoli in relazione attraverso varie tecniche crittografiche e di gestione delle informazioni.

Un albero di merkle è una struttura dati suddivisa in diversi livelli il cui scopo è metterli in relazione , nodo con un'unica radice associata ad essi. Per ottenere questo , ogni nodo è identificato univocamente attraverso un hash. Ogni nodo foglia è un hash e il suo nodo padre è l'hash della concatenazione dei nodi figli . In questo senso si arriva ad avere un root hash , che ovviamente cambia ad ogni foglia che viene aggiunta.

Quando si parla di proof merkle , in pratica il sito mirror deve restituire tutti i nodi necessari per consentire all'utente di ricalcolare la root, partendo da un nodo foglia (cosa che avviene in whatsapp). Quindi Se ad esempio abbiamo un nodo foglia X_3 , gli verranno restituiti insieme al nodo foglia dei nodi intermedi tramite i quali sarà smeplice calcolare gli altri nodi intermedi fino ad arrivare al root hash.

A livello implementativo il content provider si pre calcola la struttura dati , il server ricalcola tutta la struttura e si mantiene tutto quanto in memoria, ovviamente con vari trade off , e ogni volta che arriva una query seleziona subset di nodi da restituire. Il server che restituisce i dati non deve poter faslificare i nodi intermedi , cosa he poteva succedere in un certo senso con la vecchia verifica di whatsapp con il server che poteva sbagliare (o essere maniolato) ed inviare una chiavepubblica diversa , ffacendo cosi un MITM. Con l'uso di MHT invece il server non può falsificare nulla dato che le funzioni hash sono resistenti alle collisioni per definizione nel merkle tree.
La prova può essere fatta anche in un altro modo , con un aprova di non appartenenza. Si verifica anche nel caso in cui il server dica di non aver un determinato dato . Si dimostra usando gli elementi adiacenti , e la prova di inclusione di questi due elementi, dimostrando in questo modo che non esiste l'elemento
i_{esimo}

Paper

Allora whatsapp consente alle perosne di scambiare dei messaggi (inclusi chat groups , immagini , video) . I messaggi vocali e video chiamate tra il mittente e il

destinatario che usano il protocollo di Signal.

Un utente può avere diversi dispositivi , ognuno con il suo set di chiavi di cifratura . Se le chiavi di cifratura di un dispositivo sono compromesse , un attaccante non può usarle per decifrare i messaggi inviati ad altri dispositivi, anche dispositivi dello stesso utente. La cifratura end to end permette di che solo chi invia e chi riceve possa leggere il messaggio e nessuno nel mezzo , nemmeno whatsapp stesso.

Queste chiavi di cifratura partecipano ad un processo detto " key exchange" dove i clients scambiano le chaivi pubbliche attraverso l'infrastruttura di Whatsapp per iniziare una sessione di comunicazione con cifratura end to end con un altro utente. Verificare la correttezza di queste chiavi di sessione di cifratura però richiedono una scan del QR oppure verificando le cifre 60 digitali.

Inoltre ci vuole un assicurazione che l'utente sia effettivamente chi dice di essere , cosa che appunto avviene con lo scand el QR ma va fatto per ogni dispositivo dopo la modifica della chiave sul dispositivo principale.

Si verificano questi cambiamenti chiave.Ogni volta che l'applicazione è installata, reinstallata, registrata o liberata a un account. Questo problema si espande quadraticamente in gruppi, poiché ogni membro del gruppo ha chiavi di crittografia a coppie associate a ogni altro membro del gruppo. In un grande gruppo, in particolare, è impossibile aspettarsi che chiunque sia in grado di tenere il passo
Con la frequenza necessaria delle verifiche manuali per affermare appieno la correttezza di tutte le chiavi pubbliche in uso nel gruppo.

Fortunatamente si è pensato ad un meccanismo automatico per questo proceso e rednere piu accessibile agli utenti .

Termini -Devi e Key trasparency

- Device primario : Un device usato per registrarsi su whatsapp con un numero di telefono. Ogni account whatsapp è associato con il singolo device principale. Questo device primario può essere usato per collegarsi a dispositivi aggiuntivi.
- Public identity key: un chiave long-term ottenuta con curva ellittica Curve25519 , generata al momento dell'installazione.

- Hash digest : l'output crittografico di una funzione hash. L'implementazione del key trasparency di whatsapp usa la funzione Blake3 hash che emette 32 byte di digest binario per le operazioni di hash.
- Epoch : Il periodo di tempo per il quale è accumulato l'update delle chiavi dei client che vengono pubblicate in directory. Questo è repparesentato da un numero monolitico incrementale , detto **epoch number**, insieme ad un hash digest che impegna la directory in quel momento ,
- Merkle prefic tree : UN albero binario (0 ,1 o 2 figli) composto da foglie di informazioni dove a coppie di hash sono computate per dare il vlaore del nodo padre fino ad arrivare al root has che è alla cima dell'albero . Le foglie sono associate con delle etichtte che determina univocamente la posizione nell'abero.
- Auditable key directory : Una Open source Rust based implementaizone del key trasparency ispirato dal protocollo SEEMless con estensione adattata al Parakeet
- Publish queue : coda per raggruppare le public key identity mentre si aspetta la loro pubblicazione

Overview del sistema

In un tipico sistema di messaggistica con crittografia end to end , il service provider è responsabile di hostare una directory che contiene la mappatura degli identificativi dei contatti (numero di telefono) per ricavare la public keys. Quando l'user Alice vuole iniziare una conversazione con un altro utente Bob , il problema di Alice è quello di ottenere la chiave pubblica di Bob e soprattutto che il service provider ritorni la correttevva chiave corrispondente a Bob nella sua directory.

Nel eky trasparency invece ogni risposta alla client-initialization request, per la richeista della chive, non solo ritorna la chiave pubblica dell'utente ma anche una prova che la chiave pubblica ritornata è salvata nella directory. Questa prova è collegata all'hash digest, che è una granzia sullo stato corrente della directory e che la chiave è nella sua ultima versione se la prova è confermata.

Ci sono 4 funzioni primarie nei sistemi di key trasparency:

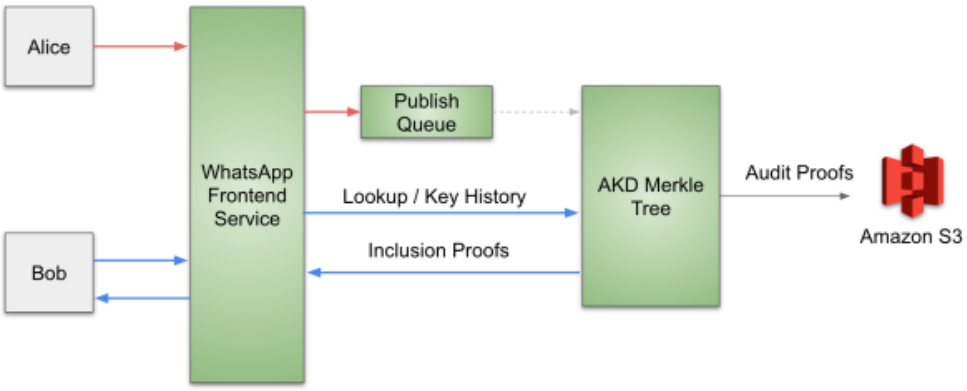
1. Registrazione device: come vengono aggiunte alla directory le nuove chiavi pubbliche, creando nuove voci oppure aggiornando quelle esistenti;
2. Lookup request (cercare la richiesta): come un client può fare check di validità della chiave che ha ricevuto da un altro utente;
3. Key history request: come un client può fare un check per la sua storia di update delle chiavi (da nogtare che la storia delle richieste non è ancora abilitata nella corrente versione del whatsapp client. Tuttavia è una funzionalità prevista per le future versioni).
4. Auditing: come il service provider può provare a degli uditors esterni che la directory è mantenuta in un modo coerente.

Il client fa due richeiste di ricerca quando un user tenta di verificare al connessione con un peer , una ricerca per la chiave di peer e un'altra per la propria chiave.

Architettura

I client si connettono con il frontend di whatsapp service in seguito a nuovi cambaimenti di chiave e quando aprono le informazioni di contatto per verificare il codice di sicurezza.

Il servizio di frontend gestisce questa connessione client ed è responsabile per interagire con altri servizi interni che rafforzano il key trasparency, includendo la coda delle chiavi pubbliche e il servizio di mantenimento del registro delle chiavi.



Il flusso di registrazione del client (linee rosse) è iniziato al momento della registrazione del client per una nuova chiave. Questa chiave è aggiunta nella coda pubblica , per essere incorporata nel commitment successivo alla prossima epoch. Dopo il contatto del client può verificare che la chiave ricevuta è nella sua versione piu recente delle chaivi di quel contatto (linee blu). Questo assicura che il client utilizza la corretta versione di cifratura per la comunicazione tra le due parti.

Whatsapp provede anche alla prova audit di consistenza per le transazioni tra epoch che sono pubblicate sull'amazon S3 che ha l'interfaccia pubblic aper tutte le entità che vogliono richiedere la prova.

Client Registration

Al momento della registrazione il client di whatsapp trasmette la chiave d'dentità pubblica al server. Il server di whatsapp salva queste chiavi d'identità pubbliche associate all'identità dell'utente. INoltre la chiave è copiata e inserita all'interno di una coda per essere inclusa nella AKD. Il flusso di eventi è cosi :

1. Il client device carica la nuova chiave di identità sul server
2. Il server salva la chiave associata all'identità dell'user e inserisce una copia nella coda pubblica
3. Ogni 5 minuti un servizio svuota la coda e genera un cmabiamento di informazioni da inserire nella directory. Questo cambiamento è sommario con il numero di versione e un nuovo hash digest root dell'albero di merkle, questo compone la nuova epoca (numero di versione e hash root).

Da notare che a collezione e l'inclusione delle chaivi d'identità pubbliche dentro AKD è fatto per tutti gli account, indipendemente dalla piattaforma.

Le informazioni codificate nelle foglie di AKD per un account contengono lo stato più recente dell'identity key public di un account per il dispositivo primario, oltre tutte le modifiche che possono essersi verificate nelle epoche precedenti fino all'epoca corrente. Questo è serializzato in binario e poi come hash , quindi digest della foglia. La posizione univoca della foglia è determianto da una funzione casuale verificabile (VRF) , calcolato sul numero dell'utente insieme al numero di versione della chiave pubblica e ad una flag che indica se la chiave è la chiave più recente per l'utente o se si tratta di una chiave precedente.

Rimozione dispositivo

Per gestire la cancellazione delle chiavi del client, impieghiamo del tombstoning nell'AKD. Questo è il processo di sovrascrittura della riga foglia con un valore costante (appunto il valore tombstone -lapide). QUesto indica che il dato er apubblicato , ma è stato rimosso e non può piu essere recuperato. Questo viene eseguito sulla rimozione del dispositivo client, sulla deregistraizone o dopo un tempo di ritenzione specificato fintanto che la chiave di identità pubblica codificata nella foglia non è più attiva.

L'hash derivato dai dati originali è ancora presente nella directory , tuttavia la sua preimagine non può più essere validata. Questo vuol dire che se il client prova a convalidare la struttura proof con una lapide presente , non può verificare che che i dati originali comportino il vlaore di hash ottenuto . La verifica client del proof è fatta senza verificare il valore grezzo. Tuttavia, le pietre tombali sono verificate per essere contigue, il che significa che una volta riscontrata una lapide, anche tutte le voci più vecchie devono essere tombate. Ciò protegge dalla tombstoning di nuove informazioni senza pulire l'intera storia dell'account oltre un determinato punto.

Rimozione del dispositivo primario

Quando il dispositivo primario è rimosso , questo innesca la una cancellazione totale dell'account. Perchè l'AKD è immutabile nella storia, questo processo lapida (tombstones) tutte le voci dell'account, comepreso la più recente.

Il processo è il seguente:

1. Il dispositivo primario richeide una cancellazione di account al server;
2. Il server schedula la cancellazione di tutte le proprietà relative all'account;
3. Tutte le versioni di stato di un account sono definite "tombstoned" nella directory.

Lookup Requests

Una lookup request (richeista di ricerca) domanda al server di generare una prova per l'ultima chiave pubblica d'identità per una determinata identità nella directory. Il controllo delle chiavi di identità pubblica per un altro dispositivo client rispetto a

un hash root afferma che il valore fornito dal server per conto dell'altro dispositivo fa parte della directory commessa dall'hash root specificato. Questo aiuta a prevenire l'equivocazione del server: vale a dire, lo scenario in cui il server tenta di fornire un valore incoerente (tra i client richiesti) per un account in un'epoca specifica.

Il processo di generazione e verifica della prova è cosi:

1. Il cliente apre entrmabi i contact info oppure la pagina di cifratura per i determinati partecipanti della chat;
2. Il client invia la richeista al server per una ricerca della prova del contatto che ha aperto con l'identificativo del contatto (numero di telefono) e la chiave pubblica che possiede per quel contatto, insieme ai suoi identificativi e all'identity public key , che consente di effettuare una self-verification dell'ultima chiave;
3. Il server check:
    a. Primo , se la chiave prevista per il contatto è pendente e aggiornata nella coda dei cambiamenti , ritorna un codice di PENDING per il client
    b. Se non è pendente, il server interroga l'AKD per generare la richiesta id ricerca della prova che gli fornisce il client
        i. Se la chiave prevista non sia trovata nell'AKD o sia un valore diverso, il server restituisce not_found indicando che il client non sarà in grado di verificare automaticamente il risultato e dovrebbe ricorrere alla verifica manuale.
4. Se è corretto , la prova è ritornata la client che deserializza e computa l'operazione hash necessario per validare la prova.
    a. Addizionalmente , il client verifica che la public identity key che riceve dal server inizialmente matchi con il contenuto della lookup proof , quindi può verificare la corretta validazione del valore della public identity key ;
    b. infine il client verifica che la firma sul root hash localmente derivaro è valida e firmata con la public key inclusa nell'applicaizone Whatsapp.
5. Il client mostra una visuale sullo stato del processo di verifica sula pagina di cifratura. Il possibile output è :

a. Verifica con successo: Il device del client ha verificato che la public identity key che ha ottenuto inizialmente dal server corrisponde a quello incluso nella AKD per entrmabi i contattti e i loro device;

b. Pending verifica: se l'account target o il proprio account ha cambiato una chiave di identità pubblica molto recentemente, è possibile che siano in attesa di inclusione nell'AKD e attualmente sono attivati. Pertanto il cliente dovrebbe provare di nuovo la verifica in seguito. Questo stato non indica che non è sbagliato nella sessione;

c. Verifica non riuscita: nel caso in cui (1) è trascorso un tempo sufficiente e (2A) la chiave di identità pubblica fornita nella prova non corrisponde alla chiave di identità pubblica fornita dal server o (2b) l'account target o il proprio account è Impossibile trovare, il dispositivo client presuppone che la sessione di crittografia end-to-end non possa essere verificata automaticamente e deve essere verificata manualmente con il contatto in questione.

Si prevede che questi step si concludano in un tempo breve nella maggior parte dei casi.

Auditing

In aggiunta alle garanzie di sicurezza date , la verifica e la correttazza continua dell'intera rete è ottenuta attraverso audit proofs. Le prove di audit, denominate anche prove o prove di coerenza, sono prove crittografiche che forniscono prove alla natura solo per le modifiche alla directory nel tempo. Affermano che in nessun momento la directory è stata ricostruita con la cronologia passata eliminata o modificata in modo irregolare.

QUeste prove sono pubbliche in modo tale che tutti possono verificarle e svolgere il ruolo degli auditor pubblici o simili. Sono disponibili qui (here) e una soluzione di auditing locale per analizzare le prove di audit e verificarle è fornita nella nostra libreria AKD open source (AKD).

Il processo per generare e verificare delle audit proof è il seguente:

1. Dopo che il processo di sequenziamento gestisce le modifiche per la prossima epoca di tempo al livello di archiviazione AKD, generiamo anche prove di audit per il periodo di tempo Epoch T a T+1 (la prossima epoca della directory).

2. Questa proof è serializzata in un fomrato compatibile backwards (protobuf) e aggiornato in un bucket AWS S3 per un uso pubblico:
    a. Il bucket S3 ha abilitato il modello WORM (Write-Once-Read-Many) con un periodo di conservazione di 5 anni. Questo aiuta a garantire che una volta che un oggetto sia scritto nel livello di archiviazione, non può essere eliminato o aggiornato per almeno 5 anni;
    b. L'accesso pubblico è previsto attraverso un portale web pubblico che AWS manteine e include un uso limitato per prevenire degli attacchi DDOS proprio verso le audit proofs
3. Ogni entità pubblica può utilizzare una libreria open source per connettersi all'endpoint pubblico e verificare le audit proofs per ogni epoca cambiata nella auditable key directory. I cambiamenti emottono un hash digest per l'ulitma epoca nell'audit proof, che può essere matchata con il root hash derivatato dai clients.

Direcotry Signatures

Dopo aver genrato un audit proof dei cambaimenti e pubblicato nella repo pubblica , il processo addizionale prevede la generazinoe di firme sul root hash della directory con la chiave privata in modo cìda essere accessibile solo a quel processo. QUesta firma rafforza AKD contro altre entità dal generare cambiamenti basati sullo stato corrente della directory,comunemente definito split-view attack, dove il server o l'attaccante forniscono una visuione differente delle chiavi dela direcotry andando a falasificare la struttura delle prove.

Questa firma afferma che il legittimo processo di sequenziamento di WhatsApp ha generato le modifiche che vengono convalidate sul lato client sotto forma di una prova di ricerca (e in futuro, una prova della cronologia chiave), senza ulteriori comunicazioni tra client e server. La chiave pubblica per questo processo di firma è inclusa nelle applicazioni binarie del client WhatsApp distribuite.

Conclusioni

Le soluzioni di trasparenza chiave possono essere distribuite mediante applicazioni di messaggistica crittografate end-to-end per consentire agli utenti di convalidare più facilmente l'autenticità delle chiavi che ricevono dal loro fornitore

di servizi. Le prove di coerenza per gli aggiornamenti chiave fungono da supplemento, ma non una sostituzione completa, alla verifica manuale esistente delle impronte digitali chiave fornite nella pagina di crittografia per un contatto.

Il nostro obiettivo con questa funzione è fornire un sistema in cui la verifica delle chiavi utente può avvenire più automaticamente, in particolare per gli utenti che non verificano manualmente i loro codici di sicurezza. WhatsApp si impegna a costruire tecnologie che salvaguardano il contenuto dei messaggi degli utenti, che include l'avvio della crittografia end-to-end su scala nel 2016, consentendo backup crittografati end-to-end nel 2021 e fornendo trasparenza chiave per la verifica della chiave automatica .

Riferimenti

1.WhatsApp Encryption Overview - technical white paper

https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf
2.About end-to-end encryption - Help center article

https://faq.whatsapp.com/820124435853543
3.About security-code change notifications - Help center article

https://faq.whatsapp.com/1524220618005378
4.Auditable key directory (AKD) implementation

https://github.com/facebook/akd
5.SEEMless: Secure End-to-End Encrypted Messaging with less trust

https://eprint.iacr.org/2018/607
6.Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging

https://eprint.iacr.org/2023/81

AKD

Overview

AKD fornisce un interfaccia per una data structure che salva la coppia chiave-valore mappato in un database verificabile. La data structure è simile ai dizionari di python, dove le voci della directory sono indicizzate da keys e consentono di memorizzare un valore con una chiave e quindi di estrarre il valore fornito dalla chiave.

Le chiavi possono anche essere aggiornate per essere associate a valori diversi. Ogni batch di aggiornamenti a questi valori-chaive le mappature sono associate a un'epoca insieme all'impegno nel database delle voci in quel momento. Il server che controlla il database può utilizzare questa libreria

Sviluppare key trasparency at whatsapp

Whatsapp ha lanciato una nuova feature di sicurezza per automatizzare la verifica delle connessioni sicure basandosi sul key trasparency. La feature non richiede delle azioni addizionali o particolari step da parte degli utenti e aiuta ad assicurare cche la conversazione sia sicura.

Il key trasparency aiuta a raffrozare la garanzia che end to end crittografia fornisce alle applicazioni di messaggistica privata e personale in modo trasparente e disponibile a tutti . Esiste come detto una libreria chiamata AKD che consente a chiunque di verifcare le prove di audit della correttezza della directory . Ciò è alla base della nostra fondamentale implementazione della trasparenza. Con directory si intende l'albero di merkle , con tutti gli hash . Tramite la AKD è possibile quindi creare una directory e interagirci come avviene nella realtà tra client e server whatsapp , interrogando ad esempio sulle epoche.

LA crittografia end to end è la base della messaggistica privata su whatsapp e contribuisce a garantire che solo tu e la persona con cui stai comunicando possiate leggere ciò che viene inviato , nemmeno whatsapp può farlo . TRa le implementazioni più utilizzate della crittografia end to end e si basa sulla crittografia a chaive pubblica sviluppata per la prima volta neglianni 70. Da un punto di vista tecnico , afficnhè la crittografia end to end sia affidabile , le

estermità (mittente e destinatario) devono sapere che le rispettive chiavi di crittografia sono autentiche e valide.

Oltre a poter verificare fisicamente il codice con lo scan del qr code oggi si usa l'alebro di merkle per creare una lbero binario che riporta tutti gli hash . La nuova funzionalità di sicurezza crittografica introdotta serve per verificare automaticamnte la connessione sicura senza la necessità si scan. per fare questo si punta sulla trasparenza delle chiavi , usando AKD, libreria open che consente ai client di verificare che la chiave di crittografia sia autentica e consentirà a chiunque di verificare le prove di controllo della correttezza della directory .

L'approccio alla trapsarenza è duplice e introduce due nuovi componenti:

1. Il server di whatsapp mantiene AKD di sola aggiunta delle chaivi pubbliche mappate agli account utente.
2. Un record di controllo di terze parti , in cui qualsiasi modifica nella directory del server viene registrata in un record di controllo disponibile pubblicamente e che preserva la privacy affinchè chiunque possa verificarlo.

Con queste due aggiunte, gli utenti potranno verificare automaticamente la sicurezza della propria conversazione grazie alla directory whatsapp. una volta implementato , gli utenti attenti alla sicurezza che utilizzano la pagina del codice di sicurezza di verifica noteranno che questo processo di verifica avviene in modo rapido e automatico.

Questo sixtema è un nuovo servizio fornito da whatsapp che si basa sul controllo pubblicoi per verificare lo stato di crittografia end to end delle conversazioni personali. Sebbene questo sistema offra strumenti di verifica facili e convenienti ai nostri utenti , coloro che desiderano verificare le proprie sessioni crittografate end to end senza utilizzare affatto i server di whatsapp sono incoraggiati ad usare il processo tradizionale di verifica del codice oltre al processo automatizzato.

Le chaivi pubbliche sono solo uno strumento a disposizione degli utenti per crittografare i propri messaggi . La chaive privata , utilizzata per decrittografare i messaggi , si trova sui dispositivi dell'utente. Nessuno , tantomento whatsapp ha accesso alle chiavi private. Un elenco di chiavi pubbliche da solo non può fornire nessun accesso ai contentuti dei messaggi, in nessun modo.

Come funziona la pagina verifica codice di sicurezza

Il punto cruciale della messaggistica crittografata end to end sono le coppie di chiavi pubblica/ privata. la chiave privata è usata per decrittografare i messaggi perosnali e non lascia mai il dispositivo. La chiave pubblica è quella che gli altri usano per cifrare i messaggi ed è quindi disponibile agli altri utenti. Questo viene fatto fornendo prima la chiave a whatsapp , che verra memorizzata e sarà fornita agli utneti che desideraranno inviare dei messaggi.

La crittografia end to end è nata per proteggersi dal person in the middle , in cui pensi di parlare con qualcuno che non è chi dice però di essere, fornendo una chiave pubblica errata in modo da poter usare la sua chiave privata per la decifratura. Per ogni contatto sappiamo esserci un qr code scannerizzabile che verifica la corrispondenza della chiave -identità. Questo è un ahsh univoco , ma c'è un problema di logistica ad esempio per i memebri di un gruppo che non sono fisicamente vicini. Inoltre le difficoltà sono diverse :

1. Difficile da realizzare in comunicazioni 1:1, soprattuto con utenti che cambiano dispositivi e quindi anche le chaivi di crittografia nel tempo
2. Ancora piu difficile nei piccoli gruppi , poi ch ogni coppia di partecipanti ha un codce univoco (non esisotno codici gruppo)
3. INoltre è quasi impossibile esibirsi in grandi gruppi . Ogni volta che qualcuno si unisce o abbadona , registra un nuovo dipositivo associato , cambia cellulare, etc, è necessario ripetere l'operazione per tutti i partecipanti.

Idealmente , questo non sarebbe un processo manuale e potrebbe essere verificato attraverso una sorta di flusso automatizzato .

Ecco che si è arrivati al key trasparency , un protocollo in cui si stabilisce una AKD su whatsapp che mantiene un registro delle modifche della chiave pubblica. Inoltre abbiamo creato un repository pubblico di terze parti di registrii delle modifiche che sono verificabili nella directory che si aggiorna ogni volta che ci sono aggiunte alla directory. Ottimo per la trasparenza e per rafforzare ulteriormente la nostra garanzia end to end. Questo conferma che le chaivi pubbliche usate da un utente per contattare un destinatario sono le stesse che usnao anche altri utenti per comunicare con il destinatario.

LA trasparenza non sostituisce lo scan del qr code . La migliora , la integra cosi :

1. La scansione del codice QR richiede due personeper coordinare la verifica fuori banda. Al contrario, la trasparenza delle chiavi richiede che un solo client

avvii ed esegua il controllo sulla directory in modo da migliorare l'accessibilità del processo di controllo;

2. La key trasparency delle chiavi funge da meccanismo di coerenza della chiave pubblica quando la verifica manuale del codice qr non è pratica
3. Serve anche come primo controllo leggero della crittografia end to end che migliora l'adozione dei controlli di crittografia end to end per piu utenti , a vantaggio della sicurezza della messaggistica generale.

Nel caso in cui il controllo automatico restituisca un risultato che indica che la connessione potrebbe essere insicura si consiglia agli utenti di procedere con il controllo manuale di verifica della sicurezza.

Key trasparency

La trasparenza della chiave descrive un protocollo in cui il server mantiene un record di sola aggiuntadella mappatura tra l'account di un utente e las ua chiave di identità pubblica. Ciò consente l agenerazione di prove di inclusione per affermare che una determinata mappatura esiste nella directory al momento dell'aggiornamento più recente.

LA key trasparency di Whatsapp si basa sui lavori accademici orginali a partire da CONIKS e SEEMless al piu recente Parakeet.

Coniks, Seemless e Parakeet

Coniks ha l'obiettivo di migliorare la sicurezza e la privacy nelle comunicazioni E2E, soprattuo nei servizi di messaggistica. Il sistema si basa sugli alberi di merkle e permette agli utenti di registrare e gestire in modo sicuro le proprie chiavi pubbliche senza doversi affidare ad altre parti.
La messsaggistica E2E è afffidabile solo se gli utenti possono recuperare le chiavi corrette. Basandosi sull'approccio CONIKS si fomralizza la nozione di Provacy preserving verifying jey direcotry , cioè un sistema che consente agli utenti di monitorare le chiavi che un servizio sta distribuendo per conto loro . Un nuovo schema VKD , migliorato in privacy e scalabilità è offerto dal protocollo SEEMLESS. Questo nuovo approccio consente ai cambiamenti chiave di entrare in vigore quasi immediatamente.

Per quanto riguarda invece parakeet , ci aiuta a descrivere un sistema di trasparenza delle chiavi progettato per migliorare la sicurezza nei sistemi di messaggistica crittografata E2EE. Per quanto riguarda parakeet quindi mira a garantire che i server di messaggistica servano chiavi pubbliche corrette agli utenti evitando attacchi MITM. Lo fa implementando VKD. Ottimizzazion delle risorse e supporto alla compattazione per ridurre le dimensioni dei dati memorizzati.

Quindi tutti e 3 i protocolli hanno caratteristiche uniche per garantire che le hiavi pubbliche siano registrate e verificate in modo trasparente e sicuro. Coniks fornisce una struttura di dati trasparente in cui gli utenti possono verificare le loro chiavi pubbliche e in whatsapp si usa per gestire l'associzione tra nomi e chiavi, in modo da permettere verificche su eventuali alterazioni da parte del server. Seemless introduce la verifiable key direcotry che fornisce dei meccanismi di verifica volti a garantire che ogni aggiornamento delle chiavi sia conforme alle regole stabilite. IN whatsapp è usato per aggiungere sicurezza alle modifiche di ogni chiave dando una tracciabilità e verifica da parte di auditor indipendenti , migliorando cosi la trasparenza. Parakeet migliora la sclabilità e l'efficienza della trasparenza delle chiavi attraverso l'ottimizzazione dello spazio di archiviazione e un protocollo di consistenza senza consenso, viene usato in whatsapp per gestire la crescita dei dati associati alle chiavi pubbliche , assicurando che il sistema rimanga sclaalbile.

Possiamo quindi dire che whatsapp key trapsarency nel suo funzionamento combina gli aspetti di questi protocolli ma li modifica su misura per un utilizzo innovativo, non si tratta quinid di una replica di questi protocolli ma si vanno ad utilizzar ei punti di forza di questi protocolli per creare il key trasparencuy.

UN esempio è la direcotory di whatsapp per il salvataggio delle chiavi che è molto simile al VKD di seemless anche se cambai in alcuni aspetti ma nella pratica è praticamente uguale, come anche il ssitema di auditor si basa su quello di seemless per alcuni aspetti.

Tutti questi aspetti sono ovviamente racchiusi nella Rust AKD di meta che permette di simulare il funzionamento dell'intera catena .

Rust e sperimentazione della AKD

Prima di passareal codice in se diamo uno sguardo alle possibili operaizoni :

1. Publishing : inserimento e aggiornamento di nuove voci nella directory
2. lookup proofs : stione delle query puntuali alla directory con prove di validità. Quindi in pratica gestisce le query (essend una prova di ricerca) sulla directory , fornendo le prove di validità basate su quelle del serever e sul root hash per ogni epoch
3. Hisotry proofs: prove per la storia degli aggionramenti di una voce. Pe run dato indice nella directory , fornisce prove per la cronologia degli aggiornamenti a questa voce , confrontando la chiave pubblica del server con quella hash root di un epoca.
4. Append only proofs: prove per dimostrare che il database si è èevoluto in modo coerente e append only. Per un paio di epoche fornsce una prova a un revisore che il database si è evoluto in modo coerente e in modalità di sola aggiutna. Queste prove di sola aggiunta utilizzano una funzione VRF per evitare di divulgare informazioni su etichetet e valori corrispondenti .

Le funzioni delle librearia sono tutte asincrone e le risposte devono essere awaited.

Un directory rappresenta un AKD . Pe rimpostare un directory dobbiamo prima impostare un DB , una configurazione ad albero e un VRF e si può usare come configurazione (già fonrito da whatasapp → WhatsAppV1Configuration che corrisponde alla configurazione utilizzata per la distribuzione della trasparenza della chiavi di whatsapp..

Le quattro operaizoni possibili sono già definite nel crate di meta ad esmepio per uqanto riguarda la pubblicazione e quindi l'aggiunta di una coppia label-value usaiamo Directory::publish. Una volta aggiunti poi ci verrà ritornato il numero della nuova epoca e l'hash root.

Nell ricerca dell aprova invece chiamiamo Directory::Lookup per generare la prova della correttezza di ricerca per un client che conferma l'esistenza di una entry. Per la verifica usiamo client::lookup_verify .

ALlo stesso tempo la sicurezza è data dalla possibilità di verifica della chiave in qaulsiasi momento . A questo fine un server che esegue un AKD deve fornire un modo per poter controllare la cronologia di una chiave. Meta ha definito nel suo

crate Drectory::key_history per dimostrare la storia dei valori di una chiave in una data epoca.

Infine l'ulitma operazione , che va oltre le chiamate API del client, l'AKD fonrisce prove ai revisori che i suoi impegni sono stati mantenuti correttamente tramite ppendOnlyProof utilizzando auditor::audit_verify.

Codice

```
use akd::{
    AkdLabel, AkdValue, Directory, EpochHash, HistoryParams, Ver
    client, auditor
};
use akd::ecvrf::HardCodedAkdVRF;
use akd::storage::memory::AsyncInMemoryDatabase;
use akd::storage::StorageManager;
use akd::errors::AkdError;
use tokio; // Assicurati che Tokio sia importato per eseguire co

type Config = akd::WhatsAppV1Configuration;

#[tokio::main]
async fn main() -> Result<(), AkdError> {
    // 1. Inizializza il database in memoria
    let db = AsyncInMemoryDatabase::new();
    let storage_manager = StorageManager::new_no_cache(db);
    let vrf = HardCodedAkdVRF {};

    // 2. Crea una nuova directory
    let akd = Directory::<Config, _, _>::new(storage_manager, vr
    println!("Directory creata.");

    // 3. **Publishing**: Inserisci nuove voci nella directory
    let entries = vec![
        (AkdLabel::from("first entry"), AkdValue::from("first va
```

```
        (AkdLabel::from("second entry"), AkdValue::from("second
    ];

    let EpochHash(epoch1, root_hash1) = akd.publish(entries).awa
    println!("Pubblicato epoch {} con root hash: {}", epoch1, he

    // 4. **Lookup Proofs**: Esegui una query e ottieni una prov
    let (lookup_proof, epoch_hash) = akd.lookup(AkdLabel::from("
    let public_key = akd.get_public_key().await?;

    let lookup_result = client::lookup_verify::<Config>(
        public_key.as_bytes(),
        epoch_hash.hash(),
        epoch_hash.epoch(),
        AkdLabel::from("first entry"),
        lookup_proof,
    )?;

    assert_eq!(
        lookup_result,
        VerifyResult {
            epoch: epoch1,
            version: 1,
            value: AkdValue::from("first value"),
        }
    );
    println!("Prova di lookup verificata con successo.");

    // 5. **History Proofs**: Pubblica una nuova versione e otti
    let entries = vec![
        (AkdLabel::from("first entry"), AkdValue::from("updated
    ];

    let EpochHash(epoch2, root_hash2) = akd.publish(entries).awa
    println!("Pubblicato epoch {} con root hash: {}", epoch2, he
```

```
    let (history_proof, _) = akd.key_history(
        &AkdLabel::from("first entry"),
        HistoryParams::default(),
    ).await?;

    let key_history_result = client::key_history_verify::<Config
        public_key.as_bytes(),
        root_hash2,
        epoch2,
        AkdLabel::from("first entry"),
        history_proof,
        akd::HistoryVerificationParams::default(),
    )?;

    assert_eq!(
        key_history_result,
        vec![
            VerifyResult {
                epoch: epoch2,
                version: 2,
                value: AkdValue::from("updated value"),
            },
            VerifyResult {
                epoch: epoch1,
                version: 1,
                value: AkdValue::from("first value"),
            },
        ]
    );
    println!("Prova di storia verificata con successo.");

    // 6. **Append-Only Proofs**: Verifica la coerenza delle mod
    let audit_proof = akd.audit(epoch1, epoch2).await?;
    let audit_result = auditor::audit_verify::<Config>(
        vec![root_hash1, root_hash2],
        audit_proof,
```

```
    ).await;

    assert!(audit_result.is_ok());
    println!("Prova append-only verificata con successo.");

    Ok(())
}
```

Quindi nel codice importiamo le dipendenze che ci servono. INizializziamo il databse e la directory

- `AsyncInMemoryDatabase::new()`: Crea un database in memoria per memorizzare le voci della directory.
- `StorageManager::new_no_cache(db)`: Crea un gestore di archiviazione senza caching.
- `HardCodedAkdVRF {}`: Usa una VRF predefinita.
- `Directory::new()`: Crea una nuova directory auditable.

Successivamente passiamo all'operazione di pubblicazione:

- `AkdLabel::from("...")` e `AkdValue::from("...")`: Creano etichette e valori per la directory.
- `akd.publish(entries).await?`: Pubblica le voci nella directory e ritorna l'`EpochHash` contenente l'epoch e il root hash.

Per la parte di lookup sfruttiamo :

- `akd.lookup(...)`: Ottiene una prova di lookup per una chiave specifica.
- `client::lookup_verify(...)`: Verifica la prova di lookup utilizzando la chiave pubblica, l'hash dell'epoch e l'epoch stesso.

Effettuiamo una modifica e una nuova pubblicazione

- `akd.publish(...)`: Pubblica una nuova versione di una voce.
- `akd.key_history(...)`: Ottiene una prova di storia per una chiave specifica.
- `client::key_history_verify(...)`: Verifica la prova di storia utilizzando il root hash e l'epoch aggiornato.

Infine la prova di append only che dimostra come le modifiche siano coerenti tra due epoch

- `akd.audit(...)`: Genera una prova di append-only che dimostra che le modifiche tra due epoch sono coerenti.
- `auditor::audit_verify(...)`: Verifica la prova di append-only utilizzando i root hash degli epoch interessati.

In sostanza con il codice sfruttiamo ilcrate di meta nel seguente modo :

- **Inizializzazione**: Crea una directory auditable in memoria.
- **Pubblicazione**: Aggiunge o aggiorna voci nella directory.
- **Lookup**: Verifica la presenza e il valore di una chiave specifica.
- **Storia**: Verifica la storia di modifiche di una chiave attraverso diversi epoch.
- **Append-Only**: Verifica che le modifiche tra due epoch siano coerenti e non abbiano rimosso voci in modo non autorizzato.

Output generato

L'output generato dal codice ci è utile soprattutto per seguire la corretta esecuzione del codice e capirne il funzionamento

1. Creazione della Directory:

   ```
   Directory creata.
   ```

   Questo vuol dire che la directory auditable è stata creata con successo in memoria utilizzando la configurazione specificata

2. Pubblicazione della prima epoca

   ```
   Pubblicato epoch 1 con root hash: 7fdfa56f34f1f892533a57c7adI
   ```

   Significa che la prima serie di voci che compaiono nel codice come "first entry" e "second entry" è stat apubblicata nella directory generando un nuovo root hash per l'epoch 1. Questo root hash è una rappresnetazione crittografica

dell’interostato della directory in quel momento (che sappiamo essere un albero di merkle)

3. Verifica del lookup

   ```
   Prova di lookup verificata con successo.
   ```

   La verifica della prova del first entry è riuscita . Questo vuol dire che il vlaore associato al first entry è stato confermato come primo valore

4. Pubblicazione del secondo epoch

   ```
   Pubblicato epoch 2 con root hash: <hash_del_secondo_epoch>
   ```

   Un nuovo valore viene pubblicato e viene aggiornata la directory e generato un nuovo root hash per l’epoch 2

5. Verfica della storia:

   ```
   Prova di storia verificata con successo.
   ```

   In questo caso avendo una storia per first entry in due versioni composdta da first entry e update value , date rispettivamente dall’epoch 1 ed epoch 2 , con questo controllo si conferma l’integrità e la correttezza della storia e delle modifiche. Nello specifico la coerenza st anel fatto che per il first entry all’inizio ad epoca 1 sia associato il first value e per epoca 2 sia asociato update value , per permettere che delle verifiche di qualsiasi tipo future possano recuperare tutti i valori precedenti della chiave.

   Infine c’è la parte di Append ony verifica , dove in sostanza si va alal ricerca dell conferma che l’evoluzione ella directory tr ail primo e il secondo epoch è stata fatta correttamente senza eliminazioni o modifiche non autorizzate. Questa directory rappresentata da un albero di merkle deve mantenere l’idea che i nuovi inserimenti non vadano a cancellare gli hash precedenti ma che appunto dal nome , append, vadano ad appenderli alla struttura esistente