DeFIRED: decentralized authorization with receiver-revocable and refutable delegations

Pieter-Jan Vrielynck
imec-DistriNet, KU Leuven
Leuven, Belgium
pieter-jan.vrielynck@kuleuven.be

Emad Heydari Beni
imec-DistriNet, KU Leuven
Leuven, Belgium
emad.heydaribeni@kuleuven.be

Kristof Jannes
imec-DistriNet, KU Leuven
Leuven, Belgium
kristof.jannes@kuleuven.be

Bert Lagaisse
imec-DistriNet, KU Leuven
Leuven, Belgium
bert.lagaisse@kuleuven.be

Wouter Joosen
imec-DistriNet, KU Leuven
Leuven, Belgium
Wouter.Joosen@kuleuven.be

ABSTRACT

A lot of research has been done over the last few years regarding decentralized authorization and access control, with existing approaches like the *WAVE framework* removing the need to rely on centralized parties for the management of access policies. However, these solutions show shortcomings regarding revocations, by not allowing *delegatees* to revoke existing and decline incoming delegations. Therefore, in this paper, we present *DeFIRED* to address this problem. DeFIRED is a decentralized authorization framework which allows its users to generate and revoke chains of resource delegations in a secure and transitive manner. Furthermore, the framework also allows the delegatees to prove that certain resources have (not) been delegated to them. Experimental results indicate that DeFIRED achieves similar performance results compared to the state of the art.

CCS CONCEPTS

• **Security and privacy** → **Authorization**; *Access control.*

KEYWORDS

Delegation, revocation, decentralized authorization

ACM Reference Format:
Pieter-Jan Vrielynck, Emad Heydari Beni, Kristof Jannes, Bert Lagaisse, and Wouter Joosen. 2022. DeFIRED: decentralized authorization with receiver-revocable and refutable delegations. In *15th European Workshop on Systems Security (EUROSEC'22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3517208.3523759

1 INTRODUCTION

Recent statistics [16] indicate that the *surface web* only represents about 4 percent of all the internet traffic around the world; the remaining 96 percent is subsumed by the *deep* and *dark web*. While the deep web represents the part of the internet which simply is not indexed by popular search engines, the dark web represents a subset of the deep web which is often associated with criminal activities like fraud, human – and drug trafficking. The dark web, home to over 50.000 extremist groups [9], allows malicious individuals to buy e.g. credit card numbers [12], passports, keyloggers and even 26 million login credentials belonging to employees at Fortune 1000 companies [11]. Furthermore, part of the deep web relies heavily on decentralized networks, making it harder for governmental institutions to take or even track them down. As an example, after being blocked by several companies like GoDaddy and Google [14], *The Daily Stormer* now leverages the IPFS peer-to-peer hypermedia platform [5] to provide their content. This allows visitors to store cached versions and redistribute copies of the website more easily.

Meanwhile, a lot of research has been done over the last few years regarding access control in decentralized settings. Existing solutions like the *WAVE framework* [4] provide *decentralized trust*, by allowing its users to both manage their permissions and delegate access to other users. We discuss the WAVE framework in more detail later. The problem with existing solutions however is that, to the best of our knowledge, none of them requires the acknowledgment of the delegatees to establish access delegations.

This allows malicious delegation issuers to associate unaware receivers with resources related to criminal activities, compromising content or undesired political views, such as the ones mentioned in the first paragraph. Furthermore, none of these solutions allow receivers to revoke existing or decline incoming delegations in a decentralized setting, nor do they allow these receivers to prove that they never accepted delegations for specific resources in the first place. While juridical procedures require a chain of proof to establish guilt, undesired associations with compromising content or criminal activity can cause serious personal image damage. e.g. on social media. In today's stream of continuous personal attacks using fake news and social media, one needs more and more the tools to disprove association with certain content, maybe even more than juridical procedures need the tools to prove it.

Therefore, this paper presents the *Decentralized File-oriented Resource Delegation Framework* (DeFIRED). Following the state of the art [4], DeFIRED is a decentralized authorization framework which allows its users to generate, revoke, prove and verify resource delegations in a secure and transitive manner. However, in contrast to other existing solutions, DeFIRED also enables delegatees to decline, revoke and disprove resource delegations. This allows

EUROSEC'22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9255-6/22/04…$15.00
https://doi.org/10.1145/3517208.3523759

users of DeFIRED to prove that they never acknowledged obtaining access to resources related to criminal activities, compromising content or undesired political views. Our evaluations show that the performance of DeFIRED is similar compared to the state of the art.

To further motivate the need for DeFIRED, we mention two additional use cases here, the *SOLID project for the decentralized web* [6] and *Medicalchain for decentralized medical ecosystems* [1]. In the SOLID project, users can store their data in so-called *Solid Pods* and delegate the data to other users by leveraging the *Web Access Control* (WAC) [8] system. However, the specification of this system uses the *Access Control List* (ACL) model to allow Solid Pod owners to delegate their resources, without explicitly asking permission to the delegatees. This allows owners to delegate access for Solid Pods containing illegal content to unaware users. Next, decentralized medical ecosystems like *Medicalchain* [1] aim on putting the patients in control of their own medical data by using blockchain technology. Medicalchain allows practitioners to request permission to read and write to permissioned medical records. Meanwhile, the patients themselves remain full control over their personal electronic health records (EHR) and the corresponding access control. However, Medicalchain only allows the patients to revoke access, by explicitly setting up a time limited gateway during the creation of the access policies. This renders it impossible for practitioners to revoke their own access and prove the revocation afterwards. This would have been a helpful feature, to allow practitioners to prove that they are not abusing their power to unnecessarily extract private information from the health records of old patients.

The remainder of this paper is structured as follows. Sections 2 and 3 provide the reader with additional background information and the threat model for our framework, respectively. Next, the system model for DeFIRED is described in section 4 on the basis of several use case scenarios. Section 5 covers the specification of the framework. Further, section 6 elaborates on the performance and the security of our framework. Finally, section 7 covers the related work, after which we conclude this paper in section 8.

2 BACKGROUND

Identity-based encryption. *Identity-based encryption* (IBE) [2] represents an asymmetric cryptography system. Every user of DeFIRED has it own *Private Key Generator* PKG^{IBE}, of which the corresponding *public parameters* PP^{IBE} may be shared across the framework. This allows other users to encrypt data according to that PKG^{IBE}, by freely choosing an *IBE ID* (a string). However, only the owner of PKG^{IBE} himself also knows the *master secret* MS^{IBE} of PKG^{IBE}. This allows the owner to generate the corresponding secret key SK^{IBE} for that IBE ID to decrypt the ciphertext.

More formally, identity-based encryption can be summarized using the following functions:

(1) $(PP^{IBE}, MS^{IBE}) \leftarrow \text{setup}()$. It generates public parameters PP^{IBE} and a master secret MS^{IBE}. PP^{IBE} is public information, but MS^{IBE} is only known to the user.
(2) $SK^{IBE} \leftarrow \text{extract}(MS^{IBE}, PP^{IBE}, ID)$. It extracts a secret key SK^{IBE}, which is unique to (PP^{IBE}, ID). Since the user owns MS^{IBE}, he is the only party who can extract SK^{IBE} from PKG^{IBE}.
(3) $c \leftarrow \text{encrypt}(PP^{IBE}, m, ID)$. It encrypts a message m using (PP^{IBE}, ID).
(4) $m \leftarrow \text{decrypt}(PP^{IBE}, c, SK^{IBE})$. It decrypts an encrypted message c using the public parameters PP^{IBE} and a secret key SK^{IBE} corresponding to the identifier ID.

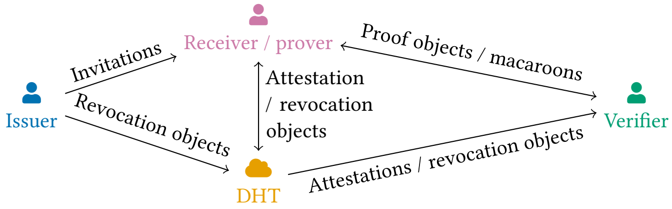
Receiver / prover
Issuer
Verifier
DHT
Invitations
Revocation objects
Attestation / revocation objects
Proof objects / macaroons
Attestations / revocation objects

Figure 1: System model of the DeFIRED framework.

Distributed hash tables. A *distributed hash table* (DHT) [13] is a distributed system for the indexing, retrieval and storage of key-value pairs among individual nodes. Both the stored data elements and network nodes have unique *identifiers* K. Distributed indexing within a DHT network is realized by forcing the individual nodes to maintain *lookup tables*, which contain references to other nodes of the network. This mechanism allows the execution of query operations for specific data, by routing the queries using UDP messages until the target node, which is responsible for hosting the queried data $DHT(K)$, is found.

Macaroons. *Macaroons* [7] are authorization credentials for controlled sharing and delegations in distributed and decentralized systems. Macaroons can be issued by individual target services, with each macaroon containing a chain of *caveats*. Caveats allow the issuers to embed predicates in the macaroons to describe policies. To prevent users from tampering with these policies, each macaroon includes a unique signature, which is calculated by recursively applying an HMAC cryptographic function along the chain of caveats, starting from a private value. Since the private values are not included in the macaroons and are therefore only known to the issuers themselves, a user can not tamper with the caveats while still maintaining a valid signature.

3 THREAT MODEL

Due to the similarities between WAVE [4] and DeFIRED, we decided to adopt a similar threat model. Therefore, adversaries are able to generate their own entities and publish them in the common DHT of our framework. Adversaries are also allowed to consult the unencrypted segments of the entities published by other users. However, adversaries should be unable to generate, revoke or modify delegations in name of *uncompromised* users. Furthermore, users should only be able to decrypt the encrypted segments of specific entities if it helps them verify/prove corresponding resource delegations. The different entities of the framework, together with the definition for a *compromised user*, will be discussed further in section 4.

4 SYSTEM MODEL

Figure 1 depicts the different entities of DeFIRED, which will be discussed in the remaining part of this section.

Users. *Users* can utilize DeFIRED to generate, revoke, (dis)prove and verify resource delegations in a non-chronological manner.

These entities are represented by the framework using *public-secret user identifier pairs* $<P\_ID\,, S\_ID>$. The *public user identifiers* P_ID are used to uniquely identify users and can be freely shared across the framework. Conversely, a user is considered compromised when his *secret user identifier* S_ID is revealed. Furthermore, each user is responsible for hosting their own REST API, to enable direct communication between different users. This tactic is often adopted in decentralized systems, e.g. in the SOLID project.

The remaining part of this paper uses several terms to refer to users, in respect to their inter-relationships. Therefore, this paper uses the term *resource owner* to refer to the owner of specific resources. Next, a user becomes an *issuer* when the user sends *invitations* to other users of the framework, the *receivers*. Finally, this paper also refers to receivers as *provers* when they send *proof objects* $P_{(\neg)Pr}$ to other users of the framework, the *verifiers*. Finally, it's the responsibility of these verifiers to verify the correctness of the received proof objects.

Invitations and attestations. To prevent issuers from issuing resource delegations without the receivers' knowledge, DeFIRED makes a clear distinction between *invitation* and *attestation* objects. Therefore, invitations are generated by the issuers and represent the issuer's admission for a specific resource delegation. On the other hand, attestations are published by the receivers in a common DHT and represent the receiver's acknowledgment of a specific invitation. In order to prevent users from forging invitations and attestations, the secret user identifiers S_ID of the issuers and receivers are used during the generation processes, respectively.

Revocations. A *revocation object*, $Revoc(S)$, is a key-value pair $<Commit(S)\,, S>$ which can be stored in the DHT of the framework. The DHT identifier (K) of a revocation object, the *revocation commitment* $Commit(S)$, represents a hashed version $Hash(S)$ of the value of the object, which is a randomly generating string. In the remainder of this paper, we refer to these randomly generated strings as *revocation secrets* S.

Both the issuers and receivers include uniquely generated revocation commitments in the invitations and attestations, respectively. This allows them to revoke existing delegations later on, by publishing revocation objects with the corresponding revocation secrets and commitments later on in the DHT.

Proof objects. The framework allows users to form chains of delegations between different users, so-called *proofs* Pr. Therefore, a single attestation may be insufficient for a prover in order to allow him to prove to a verifier that certain resources have been delegated to him. To address this issue, the concept of *proof objects* $P_{(\neg)Pr}$ is introduced to DeFIRED. Proof objects can be generated by the respective provers to prove that either certain resources *have* (P_{Pr}), or *have not* ($P_{\neg Pr}$) been delegated to them. Verifiers can afterwards fetch the necessary information from the DHT to verify the correctness of these proof objects.

Generating proof objects does *not* prevent provers from further delegating resources to other receivers of DeFIRED in the meantime. However, to improve the readability of the remainder of this paper, the assumption is made that a prover is always the receiver of the *last* attestation of a proof.

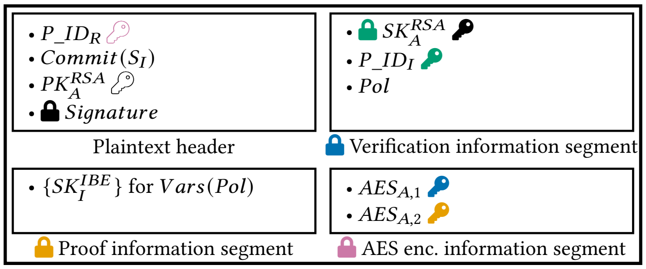

Figure 2: Illustration for the structure of the invitations.

5 USE CASE SCENARIOS DEFIRED

This section outlines the specification of DeFIRED on the basis of several scenarios.

5.1 Scenario: user generation

As mentioned in section 4, each user U of DeFIRED is represented using a unique public-secret user identifier pair $<P\_ID_U\,, S\_ID_U>$. First, the secret user identifiers $S\_ID_U\ =<SK_U^{RSA}\ ,\ MS_U^{IBE}>$ contain the master secrets MS_U^{IBE} and the secret RSA keys SK_U^{RSA}. The latter keys allow issuers and receivers to include encrypted signatures in the invitation and attestation objects, respectively. Other users can use the corresponding public keys PK_U^{RSA} from the public user identifiers $P\_ID_U\ =<PK_U^{RSA}\ ,\ PP_U^{IBE}\ >$ to verify the authenticity of these objects. Next, invitations contain confidential information about the resource delegations. To guarantee data confidentiality, DeFIRED uses identity-based encryption. Issuers can extract the public parameters PP_R^{IBE} from the public user identifiers P_ID_R of the receivers R to encrypt specific compartments of their invitations. This concept only allows users to extract the confidential information from the invitations if they know the corresponding secret IBE keys. This concept will be explained further in the specification.

5.2 Scenario: resource delegation

Establishing a complete resource delegation requires the issuer and the receiver to involve in a three-step process. These individual steps will be explained separately in the following paragraphs.

1) Generating an invitation. An invitation is generated by an issuer and consists of four compartments. The structure of the invitation objects is also illustrated by Figure 2.

The first two compartments, the *plaintext header* and the encrypted *verification information segment*, store information about the actual resource delegation. Furthermore, the verification information segment includes information which should be stored in an encrypted manner to provide data confidentiality and anonymity of the issuer I of the invitation. This includes the public user identifier of the issuer P_ID_I and the policy Pol of the delegation. This policy Pol is expressed using an existing *RTree* model [4]. This model combines the permission of the grant (*READ* or *WRITE*) together with a URI pattern describing the location of the delegated resource as in a hierarchical file system (e.g. $READ$:

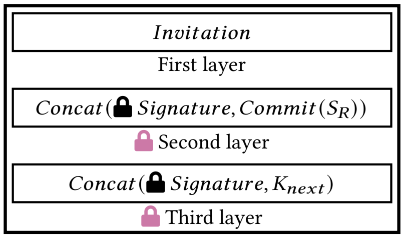
$Invitation$
First layer
$Concat($🔒$\,Signature, Commit(S_R))$
🔒 Second layer
$Concat($🔒$\,Signature, K_{next})$
🔒 Third layer

Figure 3: Illustration for the structure of the attestations.

$//data\_storage\_ID/user\_ID/shared$). On the other hand, the plaintext header contains the public user identifier of the receiver P_ID_R and a revocation commitment $Commit(S_I)$. Furthermore, to guarantee the integrity of the invitation, the plaintext header also contains a signature, which is *indirectly* encrypted using SK_I^{RSA} . The signature itself represents a hash of the invitation, *Hash(Invitation)*. This prevents malicious entities from tampering with the content of the invitation. In order to not reveal the identity of the issuer I , the signature is encrypted using an ephemeral public RSA key PK_A^{RSA} , which is also stored in the plaintext header. It's secret counterpart SK_A^{RSA} , however, is encrypted using SK_I^{RSA} and stored in the verification information segment.

The last two compartments, the encrypted *proof information* - and *AES encryption information segments*, determine which users can extract information from the encrypted segments of the invitations. More precisely, the AES encryption information segment contains the AES keys $AES_{A,1}$ and $AES_{A,2}$, which are required to decrypt the verification information and proof information segments of the invitation, respectively. The AES encryption information segment itself is encrypted using PP_R^{IBE} , with the policy *Pol* of the delegation as the corresponding IBE ID. Since the receiver R is the owner of MS_R^{IBE} , he can always generate the corresponding secret IBE key in order to obtain the AES keys. On the other hand, the proof information segment contains secret IBE keys SK_I^{IBE} generated by the issuer I , for which the policies that are *equally of less strict than Pol* are used as IBE IDs. We denote these policies as $Vars(Pol)$. Once a user can obtain these secret IBE keys, he can use them to decrypt other invitations that grant the issuer I of the invitation *the same or more* rights over the delegated resources, compared to *Pol* . This process will be explained in more depth in section 5.4.

2) Generating an attestation. The structure of the attestations is illustrated by Figure 3. The first layer of an attestation is the corresponding invitation generated by the issuer I itself. The next two layers introduce two new elements to the attestation, both generated by the receiver R : (1) another revocation commitment $Commit(S_R)$ and (2) a DHT identifier K_{next} , pointing to the next attestation in the personal queue of R . The purpose of the latter element will be discussed further in the next paragraph. Furthermore, both elements are individually concatenated with the encrypted signature from the first layer and encrypted using the secret RSA key SK_R^{RSA} from the secret user identifier S_ID_R of the receiver R . This methodology prevents other users from also tampering with the content of the last two layers of the attestation.

3) Adding an attestation to a personal queue. Receivers R organize their generated attestations in personalized queues to allow provers and verifiers to easily find the attestations published by a specific receiver. This will be explained further in sections 5.4 and 5.5.

The personal queues are implemented using linked lists. In order to host a personal queue in the DHT, receivers R publish the first attestation they generate with the hash of their public user identifier, $Hash(P\_ID_R)$, as the corresponding DHT identifier. This allows other users to easily find the heads of these personal queues, by knowing the public user identifiers of the corresponding receivers in advance. Next, receivers can add other attestations to their queues, by consecutively using the DHT identifiers K_{next} stored in the third layers of the already published attestations. Other users can easily extract these DHT identifiers from the encrypted third layers, by using the public RSA keys PK_R^{RSA} from the public user identifiers P_ID_R of the receivers R . Furthermore, these DHT identifiers can be addressed as part of a proof-of-work concept to prevent receivers from removing attestations from their queues. However, this concept will not be discussed further in this paper.

5.3 Scenario: delegation revocation

As mentioned in section 5.2, both the invitation and attestation objects contain unique revocation commitments $Commit(S)$ (= $Hash(S)$) generated by the issuers and receivers, respectively. By maintaining a map with the corresponding revocation secrets S , both users can revoke resource delegations at any time, even after the related attestations have been published. This can be achieved by generating the revocation objects $Revoc(S)$ (= <$Commit(S)$, S >) and publishing them in the DHT. It's the duty of the provers during the proof processes (section 5.4), and of the verifiers during the verification processes (section 5.5) to check, for *every* attestation of a specific proof, if a revocation object has been published for any of its revocation commitments. Finding such a revocation object during these processes infers that the specific attestation is no longer valid. The process of extracting the revocation commitments from the invitation and attestation objects, however, will be explained later on in section 5.5.

5.4 Scenario: (dis)proving delegations

As mentioned earlier in section 4, DeFIRED refers to receivers as provers when they send proof objects $P_{(\neg)Pr}$ to verifiers. These proof objects can be generated by the provers to prove that certain resources either *have* (P_{Pr}), or *have not* ($P_{\neg Pr}$) been delegated to them. The proof processes for both cases will be explained in the remaining part of this section.

Proving resource delegations. Proving a resource delegation with a policy *Pol* requires a prover to allow verifiers to reconstruct the entire proof themselves. This is done by combining the DHT identifiers and the ephemeral AES keys $AES_{A,1}$ for *every* attestation along the proof *Pr* into a single proof object P_{Pr} . Therefore, P_{Pr} is equal to $\{\{K_A , AES_{A,1}\} \mid A \in Pr\}$. This provides the verifiers with just enough information to verify P_{Pr} , a process which will be explained further in section 5.5.

Since the prover P is always the receiver of the last attestation of the proof, he can iterate over his own personal queue to find a *compatible* attestation. This is done by generating the secret IBE keys SK^{IBE} for $Vars(Pol)$ according to PKG_P^{IBE} in advance and using them to try decrypting the AES encryption information segments of the attestations. If the prover succeeds, he can obtain the ephemeral AES keys $AES_{A,1}$ and $AES_{A,2}$ of the compatible attestation. This allows him to extract the required information for P_{Pr} from that attestation and verify its authenticity and validity. The latter two processes will be explained later in section 5.5.

For the previous attestation of Pr, the verifier uses $AES_{A,1}$ and $AES_{A,2}$ to decrypt the verification information and proof information segments of the current attestation, respectively. By doing so, the prover obtains access to the public user identifier P_ID_I of the issuer I for that attestation, and the secret IBE keys SK^{IBE} for $Vars(Pol)$ according to PKG_I^{IBE}. With this information, he can iterate over the personal queue of the issuer I in order to, once again, find a compatible attestation for proof Pr. By recursively applying the previously described process up until the attestation issued by the owner of the resources is found, the prover can obtain the necessary information to construct P_{Pr}. That is, of course, as long as Pr holds.

Disproving resource delegations. Disproving a resource delegation for a policy Pol requires a prover to prove to verifiers that he never accepted a resource delegation that grants him *at least the same* permissions over the resources compared to Pol. This only requires the prover P to include the secret IBE keys SK^{IBE} for $Vars(Pol)$ according to PKG_P^{IBE} into the proof object $P_{\neg Pr}$. Therefore, $P_{\neg Pr}$ for a policy Pol is equal to $\{\{SK_{ID}^{IBE}$ according to $PKG_P^{IBE}\} \mid ID \in Vars(Pol)\}$. By using the same approach compared to the one used in the previous paragraph, verifiers can iterate over the personal queue of the prover and try to decrypt the AES encryption information segments of the attestations. If the verifier manages to find a compatible attestation and reconstruct the proof Pr himself, $P_{\neg Pr}$ does not hold.

5.5 Scenario: verifying proof objects

After the generation processes described in the previous section, provers can send their generated proof objects $P_{(\neg)Pr}$ to verifiers for verification. In case of a proof object P_{Pr}, the verifier is typically the owner of some resources, to which the prover wants to obtain access. The verification processes for both P_{Pr} and $P_{\neg Pr}$ will be discussed in the remaining part of this section.

Verifying proof objects P_{Pr}. Verifying a proof object P_{Pr} requires the verifier to (1) check the validity of the revocation commitments $Commit(S_I)$ and $Commit(S_R)$ of each attestation of the proof Pr, (2) check the authenticity of each attestation and (3) reconstruct the proof with the corresponding policies. After the verification process, the verifier can embed the resulting policy Pol of Pr into a macaroon and send it back to the prover. This way, macaroons can be used as a caching mechanism for verified proof objects, which increases the performance of the framework.

To improve the readability of this paper, we first explain how a verifier can verify proof object P_{Pr} for a proof with a single attestation. As mentioned earlier in section 5.4, P_{Pr} contains the DHT identifiers and AES keys $AES_{A,1}$ of the attestations for the proof Pr. The verifier can use that information to retrieve the corresponding attestations from the DHT and consult the content of their plaintext headers and verification information segments. Since the public user identifier P_ID_R of the receiver R is stored in the first compartment, the verifier can obtain PK_R^{RSA} to decrypt the second and third layer of an attestation. This allows the verifier to extract the revocation commitments $Commit(S_I)$ and $Commit(S_R)$ from the attestation and check their validity by consulting the DHT. Next, the verification information segment contains the encrypted, ephemeral secret RSA key ($Enc(SK_A^{RSA}, SK_I^{RSA})$). The encrypted segment can be decrypted using the public RSA key PK_I^{RSA} from the public user identifier P_ID_I of the issuer I, which is also stored in the same compartment. SK_A^{RSA} is only used to check the authenticity of the ephemeral RSA key PK_A^{RSA} from the plaintext header. In contrast, PK_A^{RSA} is used by the verifier to recalculate the encrypted signature, in order to compare it with the ones included in the plaintext header and the last two layers of the attestation. This process allows the verifier to check the authenticity of the attestation.

The verifier can repeat the previously described process for every attestation of proof Pr. To conclude the verification process, the verifier extracts the public user identifiers P_ID_I and P_ID_R, together with the policies Pol from the attestations to reconstruct the proof.

Verifying proof objects $P_{\neg Pr}$. Verifying a proof object $P_{\neg Pr}$ for a specific policy Pol only requires the verifier to verify the correctness of the included secret IBE keys SK^{IBE} for $Vars(Pol)$. The actual process of disproving a resource delegation has already been explained earlier in section 5.4.

By knowing the policy Pol in advance and extracting the IBE PKG public parameters PP_P^{IBE} from the public user identifier P_ID_P of the prover P, the verifier can encrypt random messages m using (PP_P^{IBE}, m, ID) with $ID \in Vars(Pol)$. This finally allows the verifier to verify $P_{\neg Pr}$, by checking if the secret IBE keys can be used to decrypt the encrypted messages.

6 PERFORMANCE AND SECURITY ASSESSMENT

This section covers both the performance and security assessment of the DeFIRED framework.

Performance assessment. In order to validate DeFIRED and measure its performance, we implemented the framework in Java [15] and ran several JUnit tests within a single CPX31 instance (4 vCPUs @ 2.49 GHz, 8 GB RAM, Ubuntu 20.04) on *Hetzner cloud* [10]. Furthermore, to compare the performance results with the results of the WAVE framework, the necessary *wv* and *waved* binaries were extracted from the project's GitHub page [3]. The source code of the WAVE framework, in combination with the gccgo compiler, were also used to emulate the storage layer of the framework using an in-memory storage server.

The results for the performance evaluation are shown in table 1. The first three segments of the table indicate that one can generate a new user **(1)**, delegate resources (**(2)** and **(3)**) and revoke resource delegations **(4)** with DeFIRED in 79.91, 161.4 and 0.72 milliseconds,

Operation	WAVE [ms]	DeFIRED [ms]
User generation		
(1) Generating public-secret user identifier pair	76.36	79.91
Resource delegation		
(2) Generating invitation	**(2)** and **(3)**:	160.21
(3) Generating attestation	155.90	1.19
Delegation revocation		
(4) Publishing revocation object	6.78	0.72
Proving delegations		
(5) Generating proof object P_{Pr} (single attestation)	70.07	84.90
(6) Generating proof object $P_{\neg Pr}$	/	34.13
Verifying proof objects		
(7) Verifying proof object P_{Pr} (single attestation)	24.57	26.33
(8) Verifying proof object $P_{\neg Pr}$ (single attestation)	/	83.97
(9) Generating macaroon	/	0.23
(10) Verifying macaroon	/	0.01

Table 1: Illustration of the performance results for both the WAVE framework and DeFIRED.

respectively. Next, the last two segments of the table illustrate that provers can generate proof objects P_{Pr} (**5**) and $P_{\neg Pr}$ **(6)** for a single attestation in 84.90 and 34.13 milliseconds respectively, thereby allowing provers to prove proofs with 11 consecutive attestations within a second. After the construction processes, these proof objects P_{Pr} and $P_{\neg Pr}$ can be verified by the verifiers in 26.33 **(7)** and 83.97 milliseconds **(8)**, respectively. Furthermore, the last segment of the table also shows the performance results for the construction **(9)** and verification **(10)** processes for the macaroons (0.23 and 0.01 milliseconds). These results are included in the table, to stress the performance improvement for DeFIRED by using the macaroon protocol as a caching mechanism for verified proof objects.

Finally, table 1 indicates that enabling receivers to decline and disprove resource delegations only introduces a small performance overhead to DeFIRED compared to the WAVE framework; that is, except for the revocation **(4)** and proof **(5)** processes. Publishing a revocation object *does* require the WAVE framework to add the object to an *Unequivocable Log Derived Map* transparency log [4], while DeFIRED only requires its users to host their revocation objects in the DHT. Furthermore, the verifiers of DeFIRED also checks the validity of *both* revocation commitments of each attestation during the proof processes, which explains the performance penalty for operation **(5)**.

Security assessment. On the premise that one can not derive a secret RSA or IBE key from the corresponding public elements and vice versa, DeFIRED does not allow malicious entities to forge entities in name of uncompromised users. For the invitations, forgery and tampering is prevented by including encrypted signatures in their plaintext headers. As discussed in section 5.2, these signatures are indirectly encrypted using the secret RSA keys SK_I^{RSA} from the secret user identifiers S_ID_I of the issuers I. Correctly generating the signature of a tampered invitation would therefore require a malicious entity to know S_ID_I in advance. However, as mentioned earlier in section 4, this would implicate that the issuers I of the invitations are compromised. DeFIRED prevents adversaries from forging and tampering attestations in a similar manner, by storing copies of the encrypted signatures in the encrypted second and third layers of these entities. Finally, malicious entities *can* publish revocation objects $Revoc(S)$ in the DHT, even if they are not the issuer, nor the receiver of a resource delegation. Addressing this technique to revoke specific resource delegations however, requires the malicious entities to know the revocation secrets S for at least one of the revocation commitments $Commit(S)$ stored in the corresponding attestations.

7 RELATED WORK

DeFIRED builds upon the work of the third iteration of the WAVE framework [4]. The WAVE framework represents a three-layer authorization framework offering decentralized trust. By relying on a graph-based authorization model, the framework represents each separate delegation in its *application layer* using a revocable and partly encrypted *attestation object*. Similar to DeFIRED, attestations can be chained in order to form so-called *proofs*. These attestations from the WAVE framework correspond with the invitation objects from the DeFIRED framework, and can therefore only be revoked by the issuers of the delegations. Next, the *encryption layer* of the framework provides *reverse-discoverable encryption*, a concept which is also used as part of the proof processes for DeFIRED. This allows certain users, the *provers*, to only decrypt the compartments of the attestations that are relevant to their proofs. DeFIRED addresses this concept even further, by also allowing provers to disprove resource delegations for specific policies. Finally, the *storage layer* of the WAVE framework relies on an *Unequivocable Log Derived Map* transparency log [4] for the storage of the delegation and revocation objects. Integrity is guaranteed by relying on independent auditing processes. DeFIRED eliminates these processes to improve scalability and decentralization, by using a DHT instead.

8 CONCLUSION

This paper presented DeFIRED, a decentralized authorization framework that enables delegatees to decline incoming and revoke accepted resource delegations. Furthermore, the framework allows receivers to prove and refute delegations for specific resources.

DeFIRED realizes these goals by introducing invitation and attestation objects, to represent resource delegations issued by delegation issuers and accepted by delegatees, respectively. Both types of objects can be revoked at any time. Next, the framework introduces proof objects, which can be generated by receivers (provers) to prove that certain resources have (not) been delegated to them.

Finally, DeFIRED supports chains of consecutive delegations, so-called proofs. Our performance evaluations show that the framework scales up to 11 consecutive delegations, while still allowing (1) the provers to generate and (2) verifiers to validate the corresponding proof objects in under one second.

REFERENCES

[1] Abdullah Albeyatti and Mo Tayeb. 2018. *Medicalchain*. Medicalchain. https://medicalchain.com

[2] Darpan Anand, Vineeta Khemchandani, and Rajendra Kumar Sharma. 2013. Identity-Based Cryptography Techniques and Applications (A Review). *2013 5th International Conference on Computational Intelligence and Communication Networks* (2013), 343–348. https://doi.org/10.1109/CICN.2013.78

[3] Michael P. Andersen, Sam Kumar, M. AbdelBaky, Gabe Fierro, Jack Kolb, Hyung-Sin Kim, D. Culler, and R. A. Popa. 2019. *WAVE3*. University of California, Berkeley, California, US. https://github.com/immesys/wave

[4] Michael P Andersen, Sam Kumar, Moustafa AbdelBaky, Gabe Fierro, John Kolb, Hyung-Sin Kim, David E. Culler, and Raluca Ada Popa. 2019. WAVE: A Decentralized Authorization Framework with Transitive Delegation. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1375–1392. https://www.usenix.org/conference/usenixsecurity19/presentation/andersen

[5] Juan Benet. 2015. *IPFS: Content Addressed, Versioned, P2P File System*. Protocol Labs, San Francisco, CA, US. https://ipfs.io/

[6] Tim Berners-Lee. 2016. *SOLID Project*. The W3C Solid Community Group. https://solidproject.org/

[7] Arnar Birgisson, Joe Gibbs Politz, Ulfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. 2014. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *Network and Distributed System Security Symposium*. NDSS, San Diego, California. https://doi.org/10.14722/ndss.2014.23212

[8] Sarven Capadisli. 2021. *Web Access Control specification*. The W3C Solid Community Group. Retrieved Oktober 22th, 2021 from https://solidproject.org/TR/wac

[9] Hsinchun Chen. 2012. *Dark Web: Exploring and Data Mining the Dark side of the Web*. Springer, New York. https://doi.org/10.1007/978-1-4614-1557-2

[10] Martin Hetzner. 1997. *Hetzner Cloud*. Hetzner Online GmbH, Gunzenhausen, Bavaria, Germany. https://www.hetzner.com/

[11] ID Agent 2021. *Are Your Passwords for Sale in Dark Web Markets?* ID Agent. Retrieved Oktober 22th, 2021 from https://www.idagent.com/blog/are-your-passwords-for-sale-in-dark-web-markets/

[12] Positive Technologies 2018. *The criminal cyberservices market*. Positive Technologies. Retrieved Oktober 22th, 2021 from https://www.ptsecurity.com/ww-en/analytics/darkweb-2018

[13] Ralf Steinmetz and Klaus Wehrle. 2005. *Peer-to-Peer Systems and Applications*. Springer, Berlin, Heidelberg. 79–93 pages. https://doi.org/10.1007/11530657

[14] VICE 2018. *After Crackdown, Neo-Nazis Are Hosting Propaganda on Censor-Proof Networks*. VICE. Retrieved Oktober 22th, 2021 from https://www.vice.com/en/article/43bnzd/neo-nazis-propaganda-decentralized-weev

[15] Pieter-Jan Vrielynck. 2022. *DeFIRED*. imec-DistriNet, KU Leuven. https://github.com/pvriel/Masterproef

[16] Gabriel Weimann. 2019. *Going darker? The challenge of dark net terrorism*. Wilson Center. Retrieved Oktober 22th, 2021 from https://www.wilsoncenter.org/sites/default/files/media/documents/publication/going_darker_challenge_of_dark_net_terrorism.pdf