ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20151114105745

A DISTRIBUTED AUTHENTICATION ARCHITECTURE AND PROTOCOL

Kristian Skračić, Predrag Pale, Branko Jeren

Original scientific paper

Most user authentication methods rely on a single verifier being stored at a central location within the information system. Such information storage presents a single point of compromise from a security perspective. If this system is compromised it poses a direct threat to users' digital identities if the verifier can be extracted from the system. This paper proposes a distributed authentication environment in which there is no such single point of compromise. We propose an architecture that does not rely on a single verifier to authenticate users, but rather a distributed authentication architecture where several authentication servers are used to authenticate a user. We consider an authentication environment in which the user authentication process is distributed among independent servers. Each server independently performs its own authentication of the user, for example by asking the user to complete a challenge in order to prove his claim to a digital identity. The proposed architecture allows each server to use any authentication factor. We provide a security analysis of the proposed architecture and protocol, which shows they are secure against the attacks chosen in the analysis.

Keywords: *authentication factors; digital identity; distributed authentication architecture; distributed authentication protocol; distributed user authentication*

Arhitektura i protokol za raspodijeljenu autentifikaciju korisnika

Izvorni znanstveni članak

Većina metoda autentifikacije korisnika oslanjaju se na jedan verifikator koji se pohranjuje na središnjem mjestu unutar informacijskog sustava. Takva pohrana osjetljivih informacija predstavlja jedinstvenu točku ispada iz sigurnosne perspektive. Kompromitacija verifikatora jednog sustava predstavlja izravnu prijetnju korisnikovom digitalnom identitetu. U radu se predlaže raspodijeljeno okruženje za autentifikaciju u kojem ne postoji takva točka ispada. Rad opisuje arhitekturu koja omogućuje raspodijeljenu autentifikaciju korisnika u kojoj više autentifikacijskih poslužitelja sudjeluju u provjeri autentičnosti korisnika. Razmatra se autentifikacijsko okruženje u kojem se proces autentifikacije korisnika raspodjeljuje na više nezavisnih poslužitelja. Svaki poslužitelj samostalno obavlja autentifikaciju korisnika, na primjer tražeći od korisnika da odgovori na izazov kako bi dokazao da je vlasnik digitalnog identiteta. Predložena arhitektura omogućuje svakom poslužitelju da koristi drugi autentifikacijski faktor. Provedena je sigurnosna analiza predložene arhitekture i protokola, čime se dokazuje otpornost sustava od napada odabranih u analizi.

Ključne riječi: *autentifikacijski faktori; digitalni identitet; raspodijeljena autentifikacija korisnika; raspodijeljeni autentifikacijski protokol; raspodijeljena arhitektura za autentifikaciju*

1 Introduction

User authentication presents one of the basic security requirements in information systems. Generally speaking, authentication can be described as a process in which a user offers some form of proof that he is the same user who registered the account. A proof of identity can be any piece of information that an authentication server accepts: something users have in their possession, something they know or something they are. These are called authentication factors [1]. Usually, in current practice, only one authentication server (AS) is in charge of storing the data used for authentication. When the user offers the requested proof of identity, the authentication server evaluates this proof and grants access to the user. This form of user authentication is centralized. For example, when a user tries to access his account on a typical web application he is prompted to enter a password. Traditionally, the web application holds the information about the user's account and his password. When the user submits his password during log-in process, the application compares the stored password to the submitted password. If they match, the user is granted access to the application. In other words, all the information needed to authenticate the user is held on a single system. This makes such systems the single point of compromise for securing digital identities. In other words, in case an attacker gains access to the web application, he can extract enough information to compromise the user's digital identity. Additionally, such systems often have multiple redundant copies of sensitive data [2]. By replicating confidential information across multiple servers, the risk of a successful attack becomes higher with every redundant copy [3].

We conclude there are two weaknesses in the described authentication method. First, just one piece of information is not strong enough for all applications. Second, since many users tend to use the same secret information (e.g. password) for several servers [4], revealing their identity on one compromised server, threatens their accounts on other servers with the same secret information. We believe that such an infrastructure does not offer enough security and that an attacker can significantly compromise the digital identity of a legitimate user by gaining access to the system [5]. In some current implementations, the authentication server can be completely separated from the server running web applications. For example, single sign-on schemes [6] are based on this concept. However, even in these circumstances the same security risk is present if the authentication information is stored on one server. The only difference in this scenario would be a different attack vector than with web applications.

The aim of this paper is to present a novel distributed user authentication architecture. From a security standpoint, in a distributed user authentication environment, there is no single point of compromise. The digital identities of all users of a system remain secure even in case one node is compromised. This is possible because no single node has enough information to fully authenticate a user. Instead, every node holds just a part of the needed information. We describe the required authentication information in more detail later.

The main concept in which distributed authentication differs from traditional authentication is that user authentication is no longer done on the system the user demands access to, or any other single server. Instead of authenticating the user locally, the system passes this duty to other independent systems and trusts their decision. Thus, the authentication process itself is distributed amongst two or more independent systems.

1.1 Review of distributed authentication protocols

The idea of distributed authentication is not new. There has been some research on authentication in a distributed manner (e.g., Kerberos [7]), as well as the combination of authentication and authorization in distributed systems (e.g., Keynote [8]). Distributed authentication is commonly associated with single sign-on schemes [6] or federated identity [9]. In this paper, we consider distributed authentication to be a process in which a user's claim to a digital identity is verified by two or more independent systems.

The research in [8] describes a distributed authentication model in which a single secure server is used to store secret information while other nodes carry out the processing needed to authenticate a user. An important benefit of this authentication model is that it requires no key management since the processing nodes do not handle secret data. There are many issues with achieving consistency of replicated private keys as well as achieving their secure distribution [10]. The model in [8] reduces secure server load compared to systems that run asymmetric crypto algorithms on a single server. The model is based on the SASC (Server-Aided Secret Computation) protocol, which enables a client to use the computing power of a server without revealing its secret information. Although this model reduces the risk of exposing the authentication server's private key, its performance is low compared to the conventional approaches where each node has a private key and is trusted by the secure server. This model strengthens the security of authentication systems based on public key cryptography in that it bypasses the need for key management in distributed authentication systems. However, the model is only suitable for public key based user authentication. Also, the risk of stolen identity is still present in case where the secure server is compromised.

The research in [11] introduces a new multi-server authentication scheme. The scheme is based on a registration centre, an authentication server and a smart card. A user has to register a digital identity with the registration centre, which then generates a set of parameters and stores them both locally and on a smart card. The smart cards are then delivered to the user via a secure channel. When a user wishes to authenticate with another server, he uses his smart card to compute an authentication message. The authentication server communicates with the registration centre to verify the validity of the user's authentication message and grants access to the user. This way the secret data is not stored in a central location, but is distributed across an array of smart cards. However, the cost and complexity of implementing and maintaining such a scheme may be too large for some systems.

Single sign-on has become increasingly popular, because of the increasing number of Internet services an average user uses in a day. The idea behind single sign-on scheme is that there is a single authentication server, which is used to authenticate the user for multiple services. Single sign-on schemes have an inherent flaw. They assume a service provider will trust a single third party system. The work under [12] tries to address this issue by distributing the authentication server on n different servers and using threshold cryptography. The scheme is based on sharing pieces of the authentication server's private key across a number of servers. This is achieved using a (t, n) threshold scheme with which the private key is split into n partial keys. In order for the authentication response to be valid, at least t servers have to sign it with their partial key. After at least t servers sign the response, the message appears to be signed by the authentication server's private key. This scheme aims to increase the level of security for the central authentication server. By dividing the key using a (t, n) threshold scheme the compromise of a single authentication server will not result in a compromise of the private key. Instead, an attacker would have to compromise at least t authentication servers. The downside of this scheme is that it still uses the classic username and password to authenticate users. Such an approach makes the method vulnerable to password stealing attacks. As a response to this, [12] suggests using two-factor authentication. Each user would have a unique USB device for additional authentication purposes. Such an approach makes this scheme complex and raises the overall cost of implementation and maintenance.

We conclude that existing distributed authentication methods only use predefined authentication factors and do not allow a combination of more than one factor. Therefore, any change in the authentication factor requires additional adaptation.

2 Research to define a distributed authentication architecture and protocol

We hypothesize that by using distributed authentication schemes for the very process of authenticating users, the security of digital identities can be enhanced. At the same time, we argue that a distributed authentication scheme should be cost efficient and easy to implement. This is very important from practical point of view because, as stated in [11], the marginal security benefits of any proposed mechanism may be outweighed by the complexity of implementing it, making it unusable in real word scenarios.

In this paper, we propose a new authentication scheme based on distributing the authentication process on more servers. The difference of our proposal from existing research is that we consider a distributed authentication environment with multiple nodes, in which each node holds some independent piece of information about the user that wishes to authenticate. The independent pieces of information are not bound together in any way (such as with threshold secret sharing).

The contribution of the proposed architecture is that the user's digital identity remains secure even in the case one or more of the servers are compromised. This is achieved by using an authentication method in which the authentication information is neither stored on the server the user wishes to access nor on any other single authentication server. Instead, based on the requirements described in [13], the authentication server consults with other, mutually independent, nodes (servers) that hold information relating to the user's digital identity. Each node holds just a piece of information about the user who wishes to authenticate. A user is considered authenticated only when a predefined number of nodes confirm the user's responses. Such an approach also aligns with the research in [14], which proposes a user authentication method based on vouching. Although that research is based on the premise that one user validates another's claim to a digital identity, the same principle may be applied in an environment where a claim is verified by one or more independent systems. Our contribution in this paper is focused on creating a protocol that enables secure authentication in such a distributed environment. One of the key benefits of the proposed authentication architecture is that it does not impose a specific authentication factor. Instead, each AS can choose to generate challenges based on the factor of their choosing. This is a similar process to multimodal biometric systems [15] in which two or more biometrics are used to determine the authenticity of the user. The architecture itself does not rely on cryptographic one-way functions to secure user credentials, as such solutions have been compromised in some cases [16].

We define the terminology necessary for describing the proposed authentication architecture.

- **Client** – a user that wishes to gain access to a specific service on an Application server
- **Client ID** – an identifier that is used to associate the challenges with the client. The identifier is used to identify a client in each DAS and the FAS
- **Authentication session** – defines a set of steps and servers responsible for authenticating a user once
- **Application server (AS)** - the node that the client wishes to gain access to and relies on the FAS to authenticate the user
- **Front-end Authentication Server (FAS)** – the primary node contacted by the application server, whose role is to communicate with the user during authentication. This node will forward the authentication challenges from the DAS servers to the user, and user's responses to the DAS servers as well as make final decision whether to authenticate the user or not based on decisions from the DAS servers
- **Distributed Authentication Server (DAS)** – the node that creates challenges for the user and verifies his answers
- **DAS challenge** – a question posed to the client in order to authenticate them,
- **Manifest** – a common data structure used to carry relevant information between FAS and DAS during an authentication session. The Manifest contains challenges presented by the individual DAS servers and answers given by the client.

2.1 Distributed authentication protocol

The protocol describes the behaviour of the DAS, the behaviour of the FAS, and the content of the Manifest. The protocol is independent from the challenge that the DAS generates for a user. Defining a particular authentication challenge is beyond the scope of this paper. In general, the challenge can be anything the DAS deems appropriate for the current user, using any authentication factor: biometrics, hardware tokens or any other combination thereof. The proposed authentication protocol consists of three cycles. During the first cycle the FAS gathers a number of challenges from the available DAS servers. During the second cycle the challenges are presented to the client and the client answers them. During the third cycle the answers are evaluated by the DAS servers that presented the challenge. The first and third cycles are similar. The only difference between them is that during the third cycle the Manifest contains answers instead of challenges. The following steps describe the authentication protocol. Tab. 1 and Fig. 1 explain the steps in more detail.

Table 1 Symbol explanation

Symbol	Explanation
ID_{CLIENT}	The client identifier (username or other)
ENV	The envelope
ENV_{SIG}	The envelope signature
DW_{N}	The drawer for the Nth DAS
SDW_{DASN}	Signature of the initialized drawer for the Nth DAS
EDW_{DASN}	Encrypted initialized drawer for the Nth DAS
PR_{X}	Private key operation (signature generation) using X's private key
PU_{X}	Public key operation (encryption) using X's public key
CH_{DASN}	Challenge from the Nth DAS
DWC_{DASN}	The drawer for the Nth DAS populated with a challenge for the user
CHA_{DASN}	The clients answer to the challenge of the Nth DAS
DWA_{N}	The drawer for the Nth DAS populated with the clients answer
$SDWA_{DASN}$	Signature of the drawer for the Nth DAS populated with the client answers
$EDWA_{DASN}$	Encrypted empty drawer for the Nth DAS
$VOTE_{DASN}$	The vote of the Nth DAS
DWV_{DASN}	The drawer for the Nth DAS populated with its vote
$SDWV_{FASN}$	Signature of the drawer for the Nth DAS populated with its vote
$EDWV_{FASN}$	Encrypted drawer for the Nth DAS populated with its vote

Step 1: Client → FAS:ID_{CLIENT}

The authentication session begins once the client initiates the authentication process with the FAS. Since the client is already registered with the FAS, prior to requesting access, the FAS knows which DAS servers can be used to authenticate this client. It then chooses at random a subset of the DAS servers capable of authenticating the client and compiles the manifest as described in the previous chapters. The FAS populates the Envelope of the Manifest, creates the DAS Drawers, sets the DAS operation to "generate" and digitally signs the envelope with its private key and encrypts it using its public key. Also, each DAS drawer is encrypted with the DAS's public key and digitally signed by the FAS's private key. Encryption is used to preserve confidentiality of transferred data, while the digital signature ensures Drawer authenticity and non-repudiation.

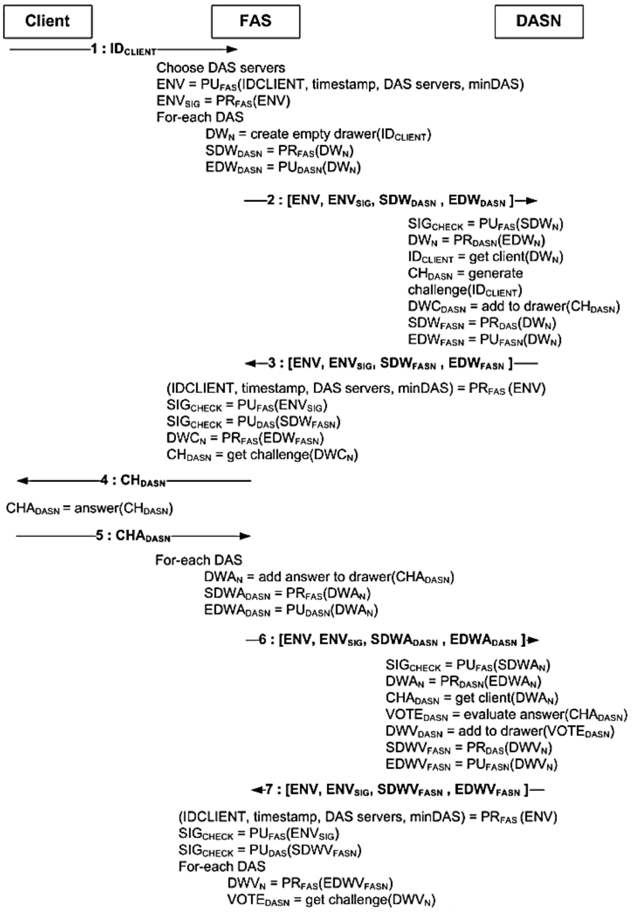
Client
FAS
DASN
1 : ID_{CLIENT}
Choose DAS servers
ENV = PU_{FAS}(IDCLIENT, timestamp, DAS servers, minDAS)
ENV_{SIG} = PR_{FAS}(ENV)
For-each DAS
DW_N = create empty drawer(ID_{CLIENT})
SDW_{DASN} = PR_{FAS}(DW_N)
EDW_{DASN} = PU_{DASN}(DW_N)
2 : [ENV, ENV_{SIG}, SDW_{DASN} , EDW_{DASN}]
SIG_{CHECK} = PU_{FAS}(SDW_N)
DW_N = PR_{DASN}(EDW_N)
ID_{CLIENT} = get client(DW_N)
CH_{DASN} = generate challenge(ID_{CLIENT})
DWC_{DASN} = add to drawer(CH_{DASN})
SDW_{FASN} = PR_{DAS}(DW_N)
EDW_{FASN} = PU_{FASN}(DW_N)
3 : [ENV, ENV_{SIG}, SDW_{FASN} , EDW_{FASN}]
(IDCLIENT, timestamp, DAS servers, minDAS) = PR_{FAS} (ENV)
SIG_{CHECK} = PU_{FAS}(ENV_{SIG})
SIG_{CHECK} = PU_{DAS}(SDW_{FASN})
DWC_N = PR_{FAS}(EDW_{FASN})
CH_{DASN} = get challenge(DWC_N)
4 : CH_{DASN}
CHA_{DASN} = answer(CH_{DASN})
5 : CHA_{DASN}
For-each DAS
DWA_N = add answer to drawer(CHA_{DASN})
$SDWA_{DASN}$ = PR_{FAS}(DWA_N)
$EDWA_{DASN}$ = PU_{DASN}(DWA_N)
6 : [ENV, ENV_{SIG}, $SDWA_{DASN}$, $EDWA_{DASN}$]
SIG_{CHECK} = PU_{FAS}($SDWA_N$)
DWA_N = PR_{DASN}($EDWA_N$)
CHA_{DASN} = get client(DWA_N)
$VOTE_{DASN}$ = evaluate answer(CHA_{DASN})
DWV_{DASN} = add to drawer($VOTE_{DASN}$)
$SDWV_{FASN}$ = PR_{DAS}(DWV_N)
$EDWV_{FASN}$ = PU_{FASN}(DWV_N)
7 : [ENV, ENV_{SIG}, $SDWV_{FASN}$, $EDWV_{FASN}$]
(IDCLIENT, timestamp, DAS servers, minDAS) = PR_{FAS} (ENV)
SIG_{CHECK} = PU_{FAS}(ENV_{SIG})
SIG_{CHECK} = PU_{DAS}($SDWV_{FASN}$)
For-each DAS
DWV_N = PR_{FAS}($EDWV_{FASN}$)
$VOTE_{DASN}$ = get challenge(DWV_N)

Figure 1 Proposed authentication scheme

Step 2: FAS → DAS_N: ENV, ENV_{SIG}, SDW_{DASN}, EDW_{DASN}

The DAS server decrypts its Drawer using its private key and checks the validity of the content with the FAS's public key. The DAS server then generates a challenge, or a number of challenges, for the client identified to the DAS server with the Client ID. Once done, the challenges are put within the Drawer and encrypted using the FAS's public key and signed using the DAS's private key. The DAS server then sends the Manifest to the next DAS server. If the next DAS does not respond, it will try to connect to the following DAS in the list, and so on until it finds a DAS that responds. In case there are no remaining DAS servers in the list or in case none of the DAS servers responds, the Manifest is returned to the FAS.

Step 3: DAS_N→ FAS: ENV, ENV_{SIG}, SDW_{FASN}, EDW_{FASN}

The FAS checks the envelope validity and the validity of the content in all DAS Drawers. The validity check is done by verifying the signatures of the Drawers and the Envelope, and decrypting them. If a decrypted Drawer is not valid it means there was a fault with the corresponding DAS server. Once the validity of the content is verified, the FAS checks if the required number of DAS servers has generated the necessary challenges. In case this condition is not met, the FAS server has to repeat the above process again with other DAS servers until the required number is met.

Step 4: FAS → Client: CH_{DASN}

The FAS forwards the challenges to the client. This can be done, one by one, sending next challenge only after the response to the previous one has been sent by the client, or in a batch, and then receiving responses in a batch, as well.

Step 5: Client → FAS: CHA_{DASN}

The client sends their answers to the FAS, one by one, or in a batch.

Step 6: FAS → DAS_N: ENV, ENV_{SIG}, $SDWA_{DASN}$, $EDWA_{DASN}$

The FAS populates DASs' Drawers with the client's answers and by setting the DAS operation to verify. Like in the first cycle, the Manifest is sent to the first DAS in the list. The DAS verifies the signature of its drawers and checks the client's answers. Based on the answer, the DAS makes a decision on the authenticity of the client and places its vote in its drawer and encrypts it using the FAS public key.

Step 7: DAS_N→ FAS: ENV, ENV_{SIG}, $SDWV_{FASN}$, $EDWV_{FASN}$

Once the Manifest returns, the Drawers and Envelope are verified like in the first cycle. If the required number of DAS servers has not voted, the authentication session starts again to fill the missing number of DAS servers (previous votes are not forgotten). Once all the necessary DAS servers have voted, the votes are evaluated and the decision is made if access will be granted to the client.

2.2 Client registration

A client has to register once with the FAS, prior to initiating an authentication request. During the registration process, the client registers a set of DAS servers he wishes to use for authentication. As presented in [17], there are many problems with creating and maintaining a global user identity. Thus, for each DAS the client must provide his ID known to a specific DAS, with which this DAS can generate a challenge for that client.

The client may choose any set of DAS servers he is registered with to authenticate him. However, from user proposed set, a subset is created consisting of DAS servers with which the FAS server has previously established relationship and has exchanged their public keys. This mode of operation was chosen as it may be more cost effective for a FAS server to only communicate with a set of DAS servers with which it has already established a trusted connection, rather than trying to establish relationship each time a user proposed some new DAS.

2.3 Modes of operation

During execution, the proposed protocol offers two types of data exchange modes to choose from. We call them: *centralized polling* and *round robin* mode. During centralized polling mode, the FAS sequentially initiates a connection with each DAS separately. This means that the list of next DAS servers is always empty. In round robin mode, the FAS sends the Manifest to the first DAS which responds (e.g. DAS1). Upon generating the challenges, the first DAS sends the Manifest to the next DAS on the list that responds, and so on. The last DAS server sends the Manifest Back to the FAS.

Such a protocol allows for configurable levels of fault tolerance and FAS load optimization during communication. The round robin mode offers higher performance but lower fault tolerance, while the centralized poling mode achieves higher fault tolerance but has lower performance. Distributing a process among multiple servers is a common method for removing performance bottlenecks in user authentication [7].

Each FAS can define its own set of rules regarding its interaction with the DAS servers which may depend on the required fault tolerance and performance characteristics. We also note that the proposed architecture has no single point of compromise. Also, the same DAS can be used in two different modes by two different FAS servers. Since the FAS server has no verifiers, its compromise cannot lead to a compromise of the user's digital identity.

2.4 Manifest structure and initialization

When the FAS receives the client's authentication request it instantiates a Manifest. Based on the desired mode of operation, the Manifest will be passed to one or more DAS servers randomly chosen from the client's list and then be sent back to the FAS. For such a dynamic communication model to work we choose to define a common data structure that will be used for exchanging data between the FAS and the DAS servers during the authentication session. As described previously, we call this structure the Manifest. In order for the communication protocol to work as described, the Manifest has to have the structure and content as shown in Fig. 2.

In order for the FAS to be sure that the Manifest was not tampered with during communication, it has to store some metadata about the authentication session. For this reason we introduce an Envelope element within the Manifest. The Envelope holds the client's ID on the FAS and the required number of DAS servers to authenticate the client. This data is necessary for the FAS to be stateless. Additionally, the Envelope holds a hashed list of all DAS servers that were supposed to be used in the authentication session. For each DAS server that is to be used during an authentication session, certain information has to be passed between the FAS and the DAS server. This data needs to be kept securely since it includes the challenges that will authenticate the users and the DAS evaluation of the client's responses. In order to prevent tampering with this information, each DAS shares a secret container with the FAS. We call this container a DAS Drawer and it is encrypted and digitally signed in order to keep integrity and confidentiality. Encryption and digital signatures are done using the RSA algorithm. Each DAS Drawer holds that DAS's ID of the client wishing to authenticate, the FAS return address, and the requested DAS operation. The DAS operation specifies which action the DAS server should perform: challenge generation or answer verification. Thus, the Drawer has a list of challenges and a field for the DAS's vote. Additionally, the Drawer holds a list of DAS servers that are next in line. Based on the chosen mode of operation, the Drawer is populated with the appropriate number of DAS servers.

The DAS vote is implemented as a Boolean variable. This is also similar to biometric authentication in which a user's biometric trait is measured and compared to a stored template [18]. Additionally, we propose that each DAS server should have its own internal scale based on which it will decide if a user is authentic or not. The final DAS votes are summed up and evaluated by the FAS. If all the votes (or a specific number of votes as suggested in [14]) are positive, the user is authenticated.

In order for a DAS to know how to access its drawer, a separate element is introduced to the Manifest, the Address book. Its purpose is to hold a list of all DAS drawers in the Manifest and their relative location within it. By accessing the Address book, a DAS can immediately retrieve its Drawer from the Manifest and attempt to decrypt it using its private key. In order to keep an attacker from identifying which DAS servers are used for a specific Client, DAS names are hashed.

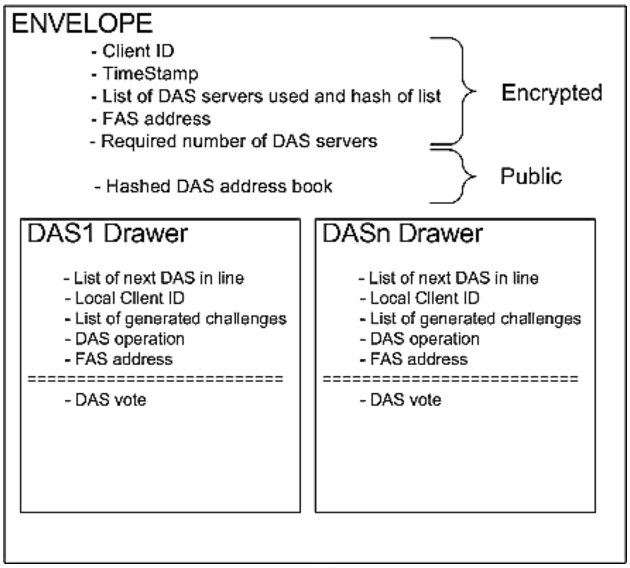

Figure 2 Manifest structure

2.5 Usability

As explained in [19], usability is a key component of each authentication system. As explained in the previous sections, the proposed system does not impose any new requirements on the client except the need for providing more than one credential during authentication. From a client's perspective, the system integrates transparently with existing authentication schemes and the complexities of the underlying mechanisms are not evident to the client. From the application server perspective, implementing the proposed authentication protocol is the same as implementing any single sign-on scheme.

Therefore, we argue that the proposed system leverages the usability of existing authentication methods and is thus as usable as the chosen authentication methods and factors.

3 Security analysis

The most important benefit of this architecture is its resistance to cyber attacks. In the following sections we show that an attacker would have to compromise all DAS servers involved in the current authentication session in order to compromise a digital identity. Additionally, even if an attacker manages to compromise all DAS servers used to authenticate one user, he is unable to steal the digital identity of another user that uses different DAS servers to authenticate with the same FAS. Also, by compromising the FAS, an attacker does not gain access to a user's digital identity since it is not stored on the FAS.

The proposed security measures are based on public key cryptography [20]. Traditionally, public key cryptography is used to establish a session key used to encrypt data with a symmetric algorithm. Such an approach is used because symmetric algorithms are faster than asymmetric ones. In the proposed protocol we only use asymmetric algorithms. Thus, no session key is established. Although they have lower performance, we argue that the use of asymmetric algorithms is not a problem for the proposed environment. The reason for this is twofold. First, the amount of data that is encrypted and decrypted is smaller than in environments that establish session keys (e.g. Web sites with HTTPS). Secondly, advances in running asymmetric algorithms in multithreaded environments [21] have been made. Also, the use of elliptic curve algorithms may additionally increase performance as described in [22].

This protocol assumes that each DAS has its own pair of private and public keys and that the FAS and the DAS servers have exchanged their public keys prior to the authentication session. This needs to be done only once between each FAS – DAS pair and will remain valid forever, or until one member of the pair changes its public-private key.

3.1 Malicious DAS

A malicious DAS attack is when one or more servers are corrupted or in collusion with an adversary. If a malicious DAS tries to change the data entered by a legitimate DAS server, the FAS will see that the Manifest was tampered with based on the digital signature in the envelope. Thus, tampering with the content will cause the authentication session to be void, and would have to be restarted. Even if a malicious DAS changes the entire Manifest content, it cannot generate a valid digital signature for a legitimate DAS server. When the FAS receives the populated Manifest it checks that all the signatures are valid. Based on this we conclude that a malicious DAS server cannot change the content of the Manifest since this will be detected. Passive attacks involving malicious DAS servers involve recording challenges and answers. During the first cycle of the protocol, the challenges are encrypted using FAS's public key. Thus, a malicious DAS cannot read the challenges as it does not know FAS's private key.

3.2 Manifest interception and manipulation

In the proposed protocol, a message can be intercepted by compromising the DAS or the FAS, or by compromising the underlying TCP/IP infrastructure (e.g. a router between two DAS servers). However, since the Manifest used for communication between the DAS servers and the FAS is secured using private key cryptography, by intercepting the Manifest the attacker can neither read its content nor falsify it.

The communication between the client and the FAS may or may not be secured. Since users usually do not pay attention to certificates and server signatures [23], a man-in-the-middle attack is possible. As described in [23], an attacker may insert himself in a secured connection between the client and the server by using a forged certificate. An attacker creates a TLS/SSL connection with the client and the server. This way, the server and the client think they are communicating with each other and the attacker can intercept their data. Currently, there is no ideal solution to this problem as it is up to the users and their browsers to detect forged certificates.

Another possible attack consists of replaying the positive votes from the DAS. If an attacker manages to record a successful authentication session and record the positive votes in the Manifest before it is sent to the FAS, the recorded messages may be sent again at a later time. For this reason we introduced a signed timestamp in the Manifest Envelope. This prevents the attacker from reusing a recorded session at a later time. Since the timestamp is generated and verified locally on the FAS, there is no need for clock synchronization amongst distributed systems. Another countermeasure is the fact that the FAS chooses the necessary DAS servers at random for each session. As described earlier, the DAS servers used for an authentication session are unknown until the FAS randomly chooses them during Manifest initialization.

We conclude that no relevant or sensitive information about the authentication session or client's digital identity can be obtained by intercepting or manipulating the Manifest.

3.3 Compromised FAS

The FAS does not keep any secret client information in its storage. Thus, its information cannot be leveraged to directly compromise a digital identity. Additionally, a compromise of one FAS does not help compromise other FAS or DAS servers as they are independent systems. However, with a compromised FAS it is possible to intercept and decrypt the Manifest content, as well as DAS Drawers since it is encrypted for the FAS. Thus, a compromised FAS could be used to record legitimate client answers to challenges. Of course, this is only a problem if the challenges repeat over time.

To prevent such an attack, we propose an additional security measure. Instead of encrypting the challenges with FAS's public key, a DAS can encrypt them using the Client's public key. Also, the Client would encrypt his answers with the DAS's public key. This way, the FAS would never know what the challenges and answers are and could not use them. As described by [24], the main design issue in authentication schemes is the choice of cryptographic algorithms and the amount of trust placed on a third party. We conclude that a compromised FAS cannot be used to compromise a client's digital identity.

4 Performance analysis of the proposed protocol

The proposed protocol relies strongly on public key cryptography. Thus, its performance relies mainly on the performance of the chosen cryptographic algorithms. In this section we give a performance analysis based on the cryptographic operations that take place in an authentication session. The notation T_{enc}, T_{dec} and T_{sig} denotes the time complexity of the encryption function, the decryption function and the digital signature function (both signature verification and generation), respectively.

4.1 Analytical approach

The proposed protocol has three major operations that are computationally intensive and that make use of cryptographic functions:

- **Manifest initialization**: the FAS has to initialize the Manifest before sending it to a DAS. The initialization occurs both times the FAS forwards the Manifest to a DAS (when soliciting challenges from DASs' and forwarding Client responses),
- **Manifest check**: FAS checks the integrity of the Manifest,
- **DAS actions**: DAS servers when they check the validity of their drawer and populate the manifest with their challenges

Tab. 2 describes the performance cost based on the number of cryptographic operations per session. Since this depends on the number of DAS server involved in an authentication session, we use the notation N to denote the number of DAS servers used.

Table 2 Cryptographic operations used

	Round robin	Centralized polling
T_{sig}	$6 + 2 * N$	$2 + 6 * N$
T_{enc}	$4 + 2 * N$	$2 + 4 * N$
T_{dec}	$4 + 2 * N$	$2 + 4 * N$

We implemented and tested the proposed protocol and architecture and measured their performance. Our performance analysis shows that average time for encryption and decryption a message of 10000 characters in memory using the RSA algorithm with key size of 512 bits is 110 milliseconds, and 70 milliseconds respectively. It was tested on a 3.20 GHz AMD Phenom II X6 CPU PC with 6.00 GB of RAM. The results we measured are similar to those in [25]. The average time for signature generation and verification using the RSA algorithm is 10 milliseconds. Thus, the average time of T_{sig} is 20 milliseconds. Our measurements are equal to those in [26]. Based on these average values we estimate the total time cost of the authentication protocol with regard to the number of DAS server (N) involved. Tab. 4 presents the time estimate for the first Cycle of the protocol in Round robin and Centralized polling mode. The time estimate for Cycle 3 is the same. We note that Centralized polling has lower performance, but offers increased security as described in previous chapters.

4.2 Practical approach

We tested the proposed architecture and protocol in a laboratory environment and measured the following results as described in Tab. 3. We implemented the system using the Java programming language and BouncyCastle library as the provider for the cryptographic functions. We choose this provider as it is commonly used to provide cryptographic functions like in existing research [27], [28], [29], [30]. As can be seen in Fig. 4, the implementation achieves greater performance than the analytical estimation when up to 10 DAS servers are used. This performance boost can be explained by the effect of the Java Just-in-Time (JIT) Compiler as described in [31] and [32].We note that for 15 and more DAS servers, the overall execution time increases exponentially. This is due to the changes in Manifest size over time. As shown in Table 3, Table 4 and Figure 3, in round robin mode the Manifest size grows with the number of DAS servers. Also, our laboratory cluster cannot efficiently manage more than 10 nodes at a time due to limitations of virtualized environments. However, we propose that 10 DAS servers are enough for most use cases. As the user has to answer 10 or more challenges from 10 different DAS servers, usability becomes an issue.

Table 3 Time estimation for one cycle of Round robin mode

N	T_{sig} (ms)	T_{enc} (ms)	T_{dec} (ms)	*TOTAL* (sec)
2	100	440	280	0,82
3	120	550	350	1,02
4	140	660	420	1,22
5	160	770	490	1,42
10	260	1320	840	2,42
15	360	1870	1190	3,42

Table 4 Time estimation for one cycle of Centralized polling mode

N	T_{sig} (ms)	T_{enc} (ms)	T_{dec} (ms)	*TOTAL* (sec)
2	140	550	350	1,04
3	200	770	490	1,46
4	260	990	630	1,88
5	320	1210	770	2,3
10	620	2310	1470	4,4
15	920	3410	2170	6,5

Based on the performance results, we stipulate that the overall time spent on cryptographic operations is lower than the time spent by the user to issue a response to the challenges presented. Additionally, the time could be lower by using elliptic curve algorithms [33]. The centralized polling mode could be further optimized if the challenges are presented to the user as they are received by the FAS. While the user is answering the challenge of the first DAS, the FAS asks the second DAS to provide a challenge, thus lowering the overall time the user is idle.

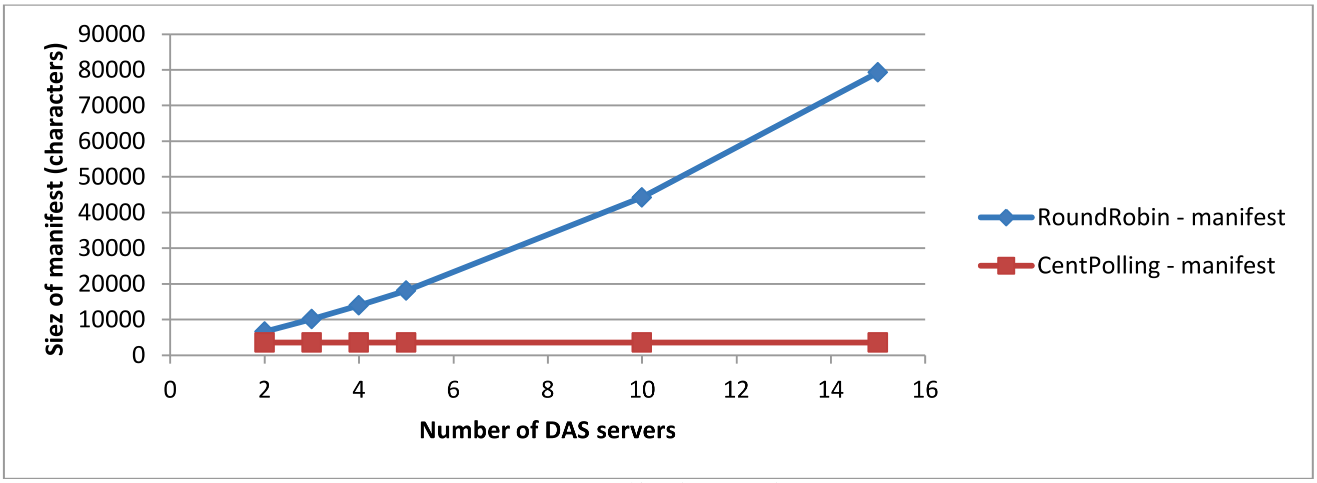

Figure 3 Manifest size comparison

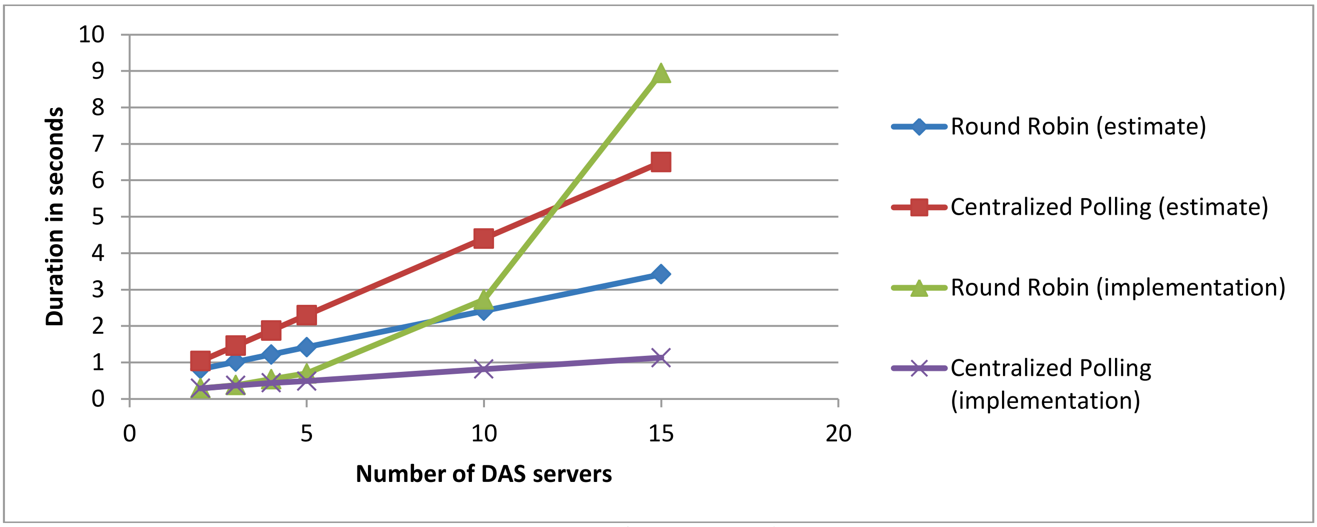

Figure 4 Performance comparison

5 Conclusion

In this paper, we proposed a new distributed user authentication architecture and protocol that allows a user to be authenticated in a distributed manner. We also analysed potential security issues and proposed countermeasures to prevent them from compromising a digital identity. The analysis has shown that the proposed distributed authentication architecture offers considerably more security than current authentication methods. Specifically, the architecture prevents a compromise of a digital identity when a single server is compromised. A key benefit of the proposed authentication architecture is that it does not impose a specific authentication factor. Instead, each DAS can use any authentication factor which is appropriate for the system. We suggest that this flexibility makes the proposed architecture and protocol impervious to change.

6 References

[1] O'Gorman, L. Comparing passwords, tokens, and biometrics for user authentication. // Proceedings of the IEEE, 91, 12(2003), pp. 2021-2040. https://doi.org/10.1109/JPROC.2003.819611

[2] Gärtner, F. C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. // ACM Comput. Surv., 31, 1(1999), pp. 1-26. https://doi.org/10.1145/311531.311532

[3] Tanaraksiritavorn, S.; Mishra, S. A Privacy Preserving Intrusion Tolerant Voting Architecture. // Network Computing and Applications, NCA 2009. Eighth IEEE International Symposium on, 9, (2009), pp. 148-155.

[4] Morris, R.; Thompson, K. Password security: a case history. // Commun. ACM, 22, 11(1979), pp. 594-597. https://doi.org/10.1145/359168.359172

[5] Herley, C.; Oorschot, P. C.; Patrick, A. S. Passwords: If We're So Smart, Why Are We Still Using Them? // in Financial Cryptography and Data Security, Springer-Verlag, 2009, pp. 230-237.

[6] Radha, V.; Reddy, D. H. A Survey on Single Sign-On Techniques. // Procedia Technology, 4, 0(2012), pp. 134-139. https://doi.org/10.1016/j.protcy.2012.05.019

[7] Liu, H.; Luo, P.; Wang, D. A distributed expansible authentication model based on Kerberos. // Journal of Network and Computer Applications, 31, 4(2008), pp. 472-486. https://doi.org/10.1016/j.jnca.2007.12.003

[8] Keromytis, A. Transport Layer Security (TLS) Authorization Using KeyNote. // Columbia University, RFC 6042, Oct. 2010.

[9] Sullivan, R. K. The case for federated identity. // Network Security, vol. 2005, 9(2005), pp. 15-19. https://doi.org/10.1016/S1353-4858(05)70283-X

[10] Hong, S.-M.; Lee, S.; Park, Y.; Cho, Y.; Yoon, H. On the construction of a powerful distributed authentication server without additional key management. // Computer Communications, 23, 17(2000), pp. 1638-1644. https://doi.org/10.1016/S0140-3664(00)00250-4

[11] Tsai, J.-L. Efficient multi-server authentication scheme based on one-way hash function without verification table. // Computers & Security, 27, 3-4(2008), pp. 115-121. https://doi.org/10.1016/j.cose.2008.04.001

[12] Brasee, K.; Kami Makki, S.; Zeadally, S. A Novel Distributed Authentication Framework for Single Sign-On Services. // Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC'08. IEEE International Conference on, pp. 52-58, 11.

[13] Skracic, K.; Pale, P.; Jeren, B. Knowledge based authentication requirements. // in 2013 36th International Convention on Information Communication Technology Electronics Microelectronics (MIPRO), 2013, pp. 1116-1120.

[14] Yesberg, J. D.; Anderson, M. S. Quantitative authentication and vouching. // Computers & Security, 15, 7(1996), pp. 633-645. https://doi.org/10.1016/S0167-4048(96)00014-4

[15] Rodrigues, R. N.; Ling, L. L.; Govindaraju, V. Robustness of multimodal biometric fusion methods against spoof attacks. // Journal of Visual Languages & Computing, 20, 3(2009), pp. 169-179. https://doi.org/10.1016/j.jvlc.2009.01.010

[16] Kumar, H. et al. Rainbow table to crack password using MD5 hashing algorithm. // Information & Communication Technologies (ICT), 2013 IEEE Conference on, pp. 433-439, Apr. 2013.

[17] Lampropoulos, K.; Denazis, S. Identity management directions in future internet. // Communications Magazine, IEEE, 49, 12(2011), pp. 74-83. https://doi.org/10.1109/MCOM.2011.6094009

[18] Jain, A.; Nandakumar, K.; Ross, A. Score normalization in multimodal biometric systems. // Pattern Recognition, 38, 12(2005), pp. 2270-2285. https://doi.org/10.1016/j.patcog.2005.01.012

[19] Weir, C. S.; Douglas, G.; Richardson, T.; Jack, M. Usable security: User preferences for authentication methods in eBanking and the effects of experience. // Interacting with Computers, 22, 3(2010), pp. 153-164. https://doi.org/10.1016/j.intcom.2009.10.001

[20] Hellman, M. An overview of public key cryptography. // Communications Society Magazine, IEEE, 16, 6(1978), pp. 24-32. https://doi.org/10.1109/MCOM.1978.1089772

[21] Dongara, P.; Vijaykumar, T. N. Accelerating private-key cryptography via multithreading on symmetric multiprocessors. // in Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software, 2003, pp. 58-69.

[22] Gupta, V.; Gupta, S.; Chang, S.; Stebila, D. Performance analysis of elliptic curve cryptography for SSL. // in Proceedings of the 1st ACM workshop on Wireless security, Atlanta, GA, USA, 2002, pp. 87-94.

[23] Callegati, F.; Cerroni, W.; Ramilli, M. Man-in-the-Middle Attack to the HTTPS Protocol. // Security & Privacy, IEEE, 7, 1(2009), pp. 78-81. https://doi.org/10.1109/MSP.2009.12

[24] Gollmann, D.; Beth, T.; Damm, F. Authentication services in distributed systems. // Computers & Security, 12, 8(1993), pp. 753-764. https://doi.org/10.1016/0167-4048(93)90041-3

[25] Fatemi Moghaddam, F.; Karimi, O.; Alrashdan, M. T. A comparative study of applying real-time encryption in cloud computing environments. // Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on, pp. 185-189, Nov. 2013.

[26] Wen-bi Rao; Quan Gan. The performance analysis of two digital signature schemes based on secure charging protocol. // Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on, vol. 2, pp. 1180-1182, Sep. 2005.

[27] Jachtoma, P.; Sakowicz, B.; Wojciechowski, J.; Napieralski, A. Application for Assigning Grades to Students using Public Key Infrastructure. // Mixed Design of Integrated Circuits and System, 2006. MIXDES 2006. Proceedings of the International Conference, pp. 773-778, 2006.

[28] Lo, J. L.-C.; Bishop, J.; Eloff, J. H. P. SMSSec: An end-to-end protocol for secure SMS. // Computers & Security, 27, 5-6(2008), pp. 154-167.

[29] Castiglione, A. et al. Engineering a secure mobile messaging framework. // Computers & Security, 31, 6(2012), pp. 771-781. https://doi.org/10.1016/j.cose.2012.06.004

[30] Vigil, M.; Buchmann, J.; Cabarcas, D.; Weinert, C.; Wiesmaier, A. Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: A survey. // Computers & Security, 50, (2015), pp. 16-32. https://doi.org/10.1016/j.cose.2014.12.004

[31] Cramer, T.; Friedman, R.; Miller, T.; Seberger, D.; Wilson, R.; Wolczko, M. Compiling Java just in time. // Micro, IEEE, 17, 3(1997), pp. 36-43. https://doi.org/10.1109/40.591653

[32] Suganuma, T. et al. Overview of the IBM Java Just-in-Time Compiler. // IBM Systems Journal, 39, 1(2000), pp. 175-193. https://doi.org/10.1147/sj.391.0175

[33] Lauter, K. The advantages of elliptic curve cryptography for wireless security. // Wireless Communications, IEEE, 11, 1(2004), pp. 62-67. https://doi.org/10.1109/MWC.2004.1269719

Authors' addresses

Kristian Skračić, mag. ing. comp.
Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Unska 3, 10 000 Zagreb, Croatia
Kristian.Skracic@fer.hr

Predrag Pale, PhD
Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Unska 3, 10 000 Zagreb, Croatia
Predrag.Pale@fer.hr

Branko Jeren, PhDFull Professor
Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Unska 3, 10 000 Zagreb, Croatia
Branko.Jeren@fer.hr