Cryptographic routing protocol for secure distribution and multiparty negotiatiated access control

Tonimir Kišasondi, Željko Hutinski
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{tonimir.kisasondi , zeljko.hutinski}@foi.hr

Abstract. *Standard key distribution protocols cannot distribute keys or content with respect to multi-party defined access rules or multi-party negotiated access or resource granting. Other possible requirements are full user anonymity or partial votes that grant access to a resource with respect to the content or key requesting side. In such systems the access rules can't be centralized and can change per host basis. We designed one such protocol that allows multi-party negotiated access control and in this work we give the suggested protocol design implementation and analysis.*

Keywords. Onion routing, secure key distribution, cryptographic protocols

1. Introduction

One of the most common requirements in corporate infrastructure is the application of multiple access control systems. For example if a client wants access to a resource like a document, a cryptographic key or any other protected element, he needs to acquire a grant for the specific element. The granting model can be simple, for example: a single password is needed to access a document, or complex like in infrastructures where we use multiple authentication elements to secure a element, like multimodal biometrics that use multiple biometric characteristics to increase the chance of identification of a person that claims to be who he represents himself to be. Our model takes the multiple element idea and expands it to another level: Trust or multiple party identification or authentication. The implementation of the protocol contains two main parts: onion routing and multiparty authentication.

2. Onion routing protocols

Onion routing protocols unlike standard routing protocols route the packet with the help of encrypted routing layers. Onion routing is extensively used in privacy and anonymity software like [5], where we use the software for anonymity and privacy. The general idea used in the [5] system is to download a list of known Tor routers, and send packets through layers.
For example, if we have a packet P and we randomly selected routers R1, R2 and R3 we would construct our packet in the following way:

$$R1\ \{R2\ [R3\ (P)]\} \quad (1)$$

Where we see, that when Router 1 receives the packet and unpacks it, he must route it to router 2 which unpacks it to send it to router 3 which finally unpacks the final packet and sends it to its destination. Also it is important to note that in this implementation each routing packet is encrypted, and the routing node can only decrypt his layer of communication so that he can know where to route the packet after decryption. That is the reason why such algorithms are called onion routing algorithms, because they work in layers. After routing in the context of anonymity, the server that received the packet would only receive the information that the packet originated from server R3, which has a plausible deniability, because he could send the data, or any other user on the Tor network [5] could have sent the same data.

From the standard architecture described above which is used by the Tor project [5], we see that the only vulnerability in the entire protocol is linked to the exit node, if the packet P is unencrypted on its way to its destination.

3. Multiple party access control, trust and verification

The next component we will use in our protocol is multiple party access control. The general idea behind the implementation is simple: considering decision makers d1, d2, d3 and d4, we want to enable access to resource R depending on the decision makers' vote. In such system there are two types of voting: unanimous, where we want all the decision makers to vote for the same, or partial voting where we need a specified amount of votes to reach a congress. In scope, decision makers can be users or network services voting for the access to data or any other resource based on trust or any other identifying measure.

In standard systems, we can use the standard 3 type identification model: "Something you know" which is usually by passwords or cryptographic keys, "Something you have" which is mostly associated with smart cards, key fobs and similar devices, or "Something you are" which is associated by biometrics. In the information systems security domain one robust and often naturally used method of identification is discarded, which is identification by a trusted mediator.

In our everyday human interaction we can be identified to a third party by a person that identified us. The same concept can be extended to information systems. Here we can suggest two approaches that we can implement. The first one is implementation with unanimous voting and the second one is implementation with partial votes that can be generalized as unanimous votes.

Both of those approaches present two ways we can approach resource granting, key distribution and content or knowledge dissemination, and other implementations that we will describe in part 5 of this paper, the implementations we discuss in detail are there just to clarify the protocol.

4. Protocol integration and description

The protocol we want to implement must adhere to certain requirements:

The right to access is defined by the decision makers where decision makers are users, end systems, network services or any combination of above. Therefore, each decision maker is a holder of a shared symmetric cryptographic key of the resource. This reasoning is based on the idea that you cannot withhold information about a resource if you exposed the user to it, because the user can write down, photograph, copy or disseminate the information in other, non computer means. Therefore if we allow a person access to a certain piece of data once, it is allowed forever until the person is cut out from the user base that can reach the information system again, when he actually cannot use any resource of the information system.

AAA ability (Authentication, Authorization and accounting) is handled by decision makers and is synchronized centrally for improved auditing and backuping. Each authorization or access is audited and can be traced.

The first variant is the implementation of a unanimous protocol. The implementation is as follows:

Assume a tuple:

{d1, d2, d3, d4... dn}

Where each d is a decision maker: a user, end system, network service or any other interested party. We see the unanimous vote as a vote where each decision maker is consent with the vote.

Our implementation requires a trusted third party, who we can identify at any time. Our suggestion is that we use PKI and LDAP infrastructure to implement a keyserver. The keyserver does not hold the resources and his identity can be verified with help of a signed certificate from a known certificate authority, and he is a repository for the knowledge about decision makers who hold the cryptographic keys to the resource. The data that the keyserver stores can be a tuple:

{r,desc,d1,d2,d3.....dn}

Where "r" is a unique identifier of a resource. Our suggestion is that the unique identifier is created by hashing the resource with two cryptographic hash functions and then appending their output for the sake of reducing collision possibility and for the sake of covertness of the transmitted data.
"desc" is a description of the resource, which is a string we can search or by which we search the base for the required resource.
"d1" to "dn" are the decision makers, which are keyholders for the resource or we can identify them with the use of keyid-s like in PGP or GPG systems. That way, we only need to trust the keyserver which is a trusted third party. The trust relationship can be enforced with digital certificates.

For the requirements of our work, we will use the following notation for the single type of protocol packet that the protocol uses:

ASE[A,X,M,Y] → Asymmetrically encrypted message M that needs to be send to A encrypted with public key of X and digital signature of the packet with secret key of Y. Also any other applicable cryptographic algorithm can be used like [4] or [1] but we used the ASE shorthand in our example, because the algorithm depends on the implementation.

The unanimous protocol is as follows:

1. Client Cl checks the authenticity of Ks1
2. Client Cl sends the message to Ks1:
ASE[Ks1,Ks1,(Need R),Ds(Cl)]
3. Keyserver Ks1 verifies the identity of the client Cl, if it is valid it continues
4. Ks1 looks up his database, finds the keyholders Kh1,Kh2 and Kh3
5. Ks1 creates a 2048 bit random sequence Q, and stores it in a base as a tuple {Q,Cl,R,t}, where t is the time of the requesting. Q is used as a nonce to protect Ks1 against packet replay or packet forgery attacks
6. Ks1 pings each Kh to see if their hosts are online
7. Ks1 checks if each Kh is able to interpret messages (listener check)
8. Ks1 Constructs a packet:
ASE[Kh1,Kh1,(Auth Cl for access on R?;
ASE[Kh2,Kh2,(Auth Cl for access on R?;
ASE[Kh3,Kh3,(Auth Cl for access on R?;
ASE[Ks,Ks,(Q) , Ds(Ks1)]) ,Ds(Ks1)]),
Ds(Ks1)]), Ds(Ks1)]
9. The packet is routed to Kh1
9.1. Kh1 verifies the authenticy of the packet
9.2. If Kh1 accepts Cl1's access to R he routes the packet to the next node (Kh2)
9.3. If he rejects Cl's access to R he sends a message to Cl1: ASE[Ks1,Ks1,(Deny Cl to R), Ds (Kh1)]
And drops the packet, effectively breaking the authorization chain.
10. Each of the other nodes that receive a packet act in the same manner
11. When a keyserver receives a packet that is routed for him, he decrypts it and compares Q to his Q in the database, and if they match he finds a random Kh, and constructs and sends the following packets:
ASE[Kh2,Kh2,(Accept Cl),Ds(Ks1)]
ASE[Cl,Cl,(Get key from Kh2),Ds(Ks1)]
Also, he adds Cl as a keyholder in the database
12. Cl can then receive the key from Kh2 and access the resource R

The following implementation is quite robust, the difficulty of subverting it in the best case depends on the key factorization of the keyholder key. That way we differentiate security from the server side, into the client side where it should actually matter.

The partial voting algorithm is similar to the previous algorithm, but has a different packet creation design. In this variant, we only need a partial number of votes to allow access to a resource,

1. Client Cl checks the authenticity of Ks1
2. Client Cl sends the message to Ks1:
ASE[Ks1,Ks1,(Need R),Ds(Cl)]
3. Keyserver Ks1 verifies the identity of the client Cl, if it is valid it continues
4. Ks1 looks up his database, finds the keyholders Kh1,Kh2 and Kh3
5. Ks1 creates multiple 2048 bit random sequence Q1, Q2 and Q3, and stores it in a base as a tuples.
6. Ks1 pings each Kh to see if their hosts are online
7. Ks1 checks if each Kh is able to interpret messages (listener check)
8. Ks1 constructs three packets, each for one Kh:
ASE[Kh1,Kh1,(Auth Cl for access on R? ;
ASE[Ks1,Ks1,(Q1),Ds(Kh1)],Ds(Kh1)]
ASE[Kh2,Kh2,(Auth Cl for access on R? ;
ASE[Ks1,Ks1,(Q2),Ds(Kh1)],Ds(Kh2)]
ASE[Kh3,Kh3,(Auth Cl for access on R? ;
ASE[Ks1,Ks1,(Q3),Ds(Kh1)],Ds(Kh3)]

9. The packet is routed to Kh1
9.1 Kh1 verifies the authenticy of the packet
9.2 If Kh1 accepts Cl1's access to R he routes the packet to the KS
9.3 If he rejects Cl's access to R he sends a message to Cl1: ASE[Ks1,Ks1,(Deny Cl to R),Ds(Kh1)]
And drops the packet, effectively the keyserver looses one vote from that person.
10. Each of the other nodes that receive a packet act in the same manner
11. When a keyserver receives a packet that is routed for him, he decrypts it and compares the recieved Q to his Q in the database, and counts the votes. If the votes exceed a specified threshold, he finds a random Kh, and constructs and sends the following packets:
ASE[Kh3,Kh3,(Accept Cl),Ds(Ks1)]
ASE[Cl,Cl,(Get key from Kh3),Ds(Ks1)]
12. Also, he adds Cl as a keyholder in the database
13. Cl can then receive the key from Kh2 and access the resource R

The most complex part of the implementation depends on how does the keyholder decides that he wants to grant access to a resource. The first and most simple implementation is to create a simple application that asks the user if he wishes to grant that access. This is more involving for the keyholder and is recommended in areas that require high security. Our recommended way is to implement a decentralized security policy, therefore if someone is allowed an resource access, only one keyholder must deny the user the access to the resource. That way, we can instantly enforce an access policy without the additional haste or complications with syncronization, which can be done for example: once a day.

5. Suggested protocol implementation

In the shown protocols, it is evident that the unanimous protocol offers better benefits over standard protocols or protocols like the partial voting protocol described above. The most important benefit is the traffic requirement and state tracking requirement. The unanimous protocol requires n+2 transmissions where a normal centralized protocol would require at least a minimum of 2n transmissions. This is important in distributed environments where we have different link types and we simply want to keep as little communication overhead and state saving as possible. The onion routing scheme enables this benefit.

Distributed auditing can be trivially achieved if each client and the keyserver keep their audit trails and synchronize them with the keyserver or any other auditing server for quick validation. Of course, the centralized auditing is optional, but all clients keep their auditing trails and can see exactly who gave access to who and to which resource was the access granted.

The partial voting algorithm can be generalized into a unanimous voting protocol if we reduce the routing layers to 1. Therefore, we create multiple two layer packets instead of one multiple layer packet. With that in mind, we show that each partial voting problem can be generalized as a multiple unanimous voting problem and that with the generation of such packets we can create any number of combinations that accommodate to our authorization needs.

This gives us the idea that the noted algorithms can be used perfectly for key management or content distribution in terms of knowledge or content management. Assuming each resource is protected with a random generated cryptographic key, we can use the unanimous algorithm to handle the key distribution tasks. The security of the entire protocol is based on the cryptanalytic difficulty of the used symmetric and asymmetric algorithms that are chosen in the implementation. We suggest the use of AES256 and 384-bit prime modulus for ECC per NSA suite B cryptographic recommendations for Top Secret level traffic. [9]

6. Conclusion

In this work we have shown a cryptographic routing algorithm for key distribution or resource granting. Additional research would include implementation of the algorithm with [6] or [7]. That way a transparent authentication framework can be implemented with good reliability. Also, additional implementations with webs of trust and keysigning protocols can be sought which would prove even better trust relationships, but with greater overhead.

Our main focus was to develop a lightweight, simple protocol that would require as little communication as possible. Our future research will be focused on implementing this algorithm as a overlay for web of trust implementation like in [8]. That way, we could improve the access control system with a distributed asymmetric key system.

7. References

[1] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", IEEE Transactions on Information Theory, v. IT-31, n. 4, 1985, pp469–472

[2] L. Eschenauer, V.D. Gligor: A key-management scheme for distributed sensor networks Proceedings of the 9th ACM conference on Computer and and Communications Security, 2002

[3] M. Hooks, J. Miles, F. Css, P. Reynolds, O. Astrachan: Onion routing and online anonymity, 2006

[4] R. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 21 1978.

[5] http://www.torproject.org/ (Acc: 11.05.2009)

[6] http://www.kernel.org/pub/linux/libs/pam/ (Acc: 11.05.2009)

[7] http://www.openldap.org/ (Acc: 11.05.2009)

[8] http://www.gnupg.org (Acc: 11.05.2009)

[9]http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml (Acc: 11.05.2009)