COMPUTER SECURITY-SUMMER SESSION-2, 1996
STEPHANE G. BOUNIOL
FULL-TIME GRADUATE STUDENT
ID 147-98-2390

THE PUZZLE THEOREM

THE LESS I KNOW, THE LESS I CAN DISCLOSE...

1. ABSTRACT

This paper presents an architecture designed to improve the efficiency of secure protocols by apportioning keys. It allows to handle with "partially trusted" hosts set in a networking environment. It is also designed to manage complex keys with "*virtual smartcards*". Access through several objects(users, processes...) requires a specified amount of trusted severs before being granted. It is based on the notion of "key apportion" or "fragmentation and dissemination"[iii]. This architecture is applied to the KERBEROS protocol to solve the problem of storing all information at one sensible point: the Key Server.

Stephane G. BOUNIOL, sbouniol@menger.eecs.stevens-tech.edu,
Phone: (UNIX-SA 9:00am-4:00pm) (212) 892 6864, Home (201) 659 5212
PO-Box: S-1622, Stevens Institute of Technology, Castle Point on Hudson,
Hoboken, NJ 07030, USA

2. INTRODUCTION

While waiting for a new version of IP, maybe more secure, the raising interest of Networking environment like Internet has considerably increase the security requirements of distributed architectures. This area of research has led to firewall technologies[i], but also to distributed protocols like KERBEROS.

KERBEROS deals with passive sniffing attacks, however several weakness have been identified[ix]. This paper's goal is to improve the security of the KERBEROS protocol where all sensible information are stored on one server[1]. But "Key Apportion" can be used in many other kind of distributed security systems.

2.1 PREVIOUS WORKS

Key distribution is an important research area. Almost all distributed security systems have raised their own way to distribute a key on several sites. However, they distribute keys in such way that each site possess a whole key. The idea relevant to this project is to "apportion"[2] a key on a whole (distributed) system, each site having only a part of the key.

One of the goal of this Protocol is to design "Limited-Intrusion Tolerant" systems. An interesting line of investigation, therefore, was to study "Fault Tolerant" systems[ii] and auto-corrective codes. Nevertheless, this paper doesn't study Byzantine[iii] faults tolerance.

The ideas presented in this paper is very close to the works developed by a French Laboratory: (LAAS)[iv,v]. They have already implemented a distributed File System, a distributed security server, and several other distributed applications. They apportion keys and replicate them in such a way that the whole information is always available. This last point isn't a requisite for an application to KERBEROS. Although, to not implementing it, can sensibly simplify the modification of the actual protocol, and greatly reduce the size of the data required.

2.2 ASSUMPTIONS

The reader should be familiar with the KERBEROS[vi,vii,viii,ix] protocol in order to look through this paper.

[1] In fact, Realms features and backups servers share theses information, However if one server is corrupted then the whole system is endangered.

[2] To avoid any confusion, I try to use a word different from "distribute". In their works, LAAS team prefers to use the terms of "Fragmentation, Dissemination and Redundancy".

3. THE PROTOCOL AND ITS ANALYSIS

The starting point of the protocol developed here is describe in the "Puzzle Idea" section. Then the "Key Apportion Model" is applied on KERBEROS. And last, the protocol is evaluated and its possible enhancement presented.

3.1 THE PUZZLE IDEA

While reading a Book designed like a Puzzle (a crossing of **Hoffstadter** "Goedel, Escher & Bach" and **Balzac**'s works[x]), I get the following Idea: In a well designed puzzle, it's impossible to deduce any useful information about the overall picture with only a single piece. However, with enough pieces (even without all of them), one can easily get an idea of the picture. Let's assume that in the context of distributed security algorithms, the puzzle pieces are some portions of a key…

3.2 THE KEY APPORTION MODEL

(Adapted to KERBEROS)

After reminding the global architecture of KERBEROS, an example of Key apportions is given, then the formal Model is presented.

3.2.1 DEFINITIONS:

n: number of servers

p: minimum number of trusted servers to allow access

$\mathbf{K}_{\mathbf{name}_I}$: Key of **name** index **I**

WS: Work Station(KERBEROS Client)

KKDS: KERBEROS Key Distribution Server

{inf}K : message **inf** encrypted with key **K**

|K|: size of the Key **K**

3.2.2 ASSUMPTION:

p<=n

3.2.3 SIMPLIFIED SCHEME OF KERBEROS PROTOCOL

The KERBEROS protocol implements many features not represented in the following scheme. It allows the definition of Realms, uses Time Stamps, and Authentication delegation via KERBEROS Tickets Granting Services. All these elements, have no direct incidences on the "key apportion" model. To enhance clarity I have simplified as much as possible the scheme.

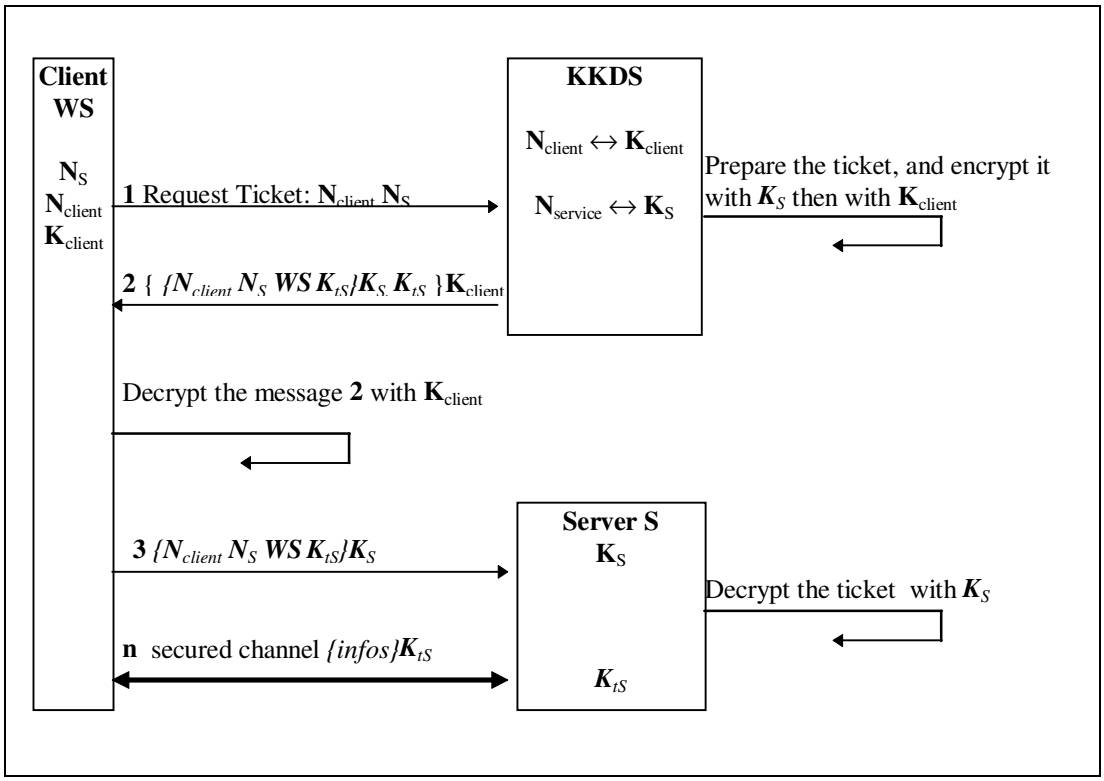


Figure 1 - Simplified KERBEROS Protocol

1) The client requests a Ticket for a service S to the KKDS(KERBEROS Key Distribution Server). The Work station knows its name and the name of the service. The message is in cleartext on the network.

2) The KKDS looks up N_{client} and N_S, finding the private keys K_{client} and K_S. It prepares the Ticket: $\boldsymbol{N_{client}\ N_S\ WS\ K_{tS}}$, enciphered it with $\boldsymbol{K_S}$. Where $\boldsymbol{K_{tS}}$ is a temporary key used for the secure channel. Then it sends the following message enciphered with $\mathbf{K}_{client}$ to the client:

$$\{\ \{\boldsymbol{N_{client}\ N_S\ WS\ K_{tS}}\}\boldsymbol{K_S}, \boldsymbol{K_{tS}}\ \}\mathbf{K}_{client}$$

3) The Work Station decrypts the message using the $\mathbf{K}_{client}$ password. Then send the ticket $\{\boldsymbol{N_{client}\ N_S\ WS\ K_{tS}}\}\boldsymbol{K_S}$ to the server S and store $\boldsymbol{K_{tS}}$ for further communication with S.

4) The Server S decrypts the ticket using its Key $\boldsymbol{K_S}$, and store $\boldsymbol{K_{tS}}$ for further communication with the Work Station.

3.2.4 SIMPLE SCHEME OF AN EXAMPLE OF KERBEROS WITH KEY APPORTION PROTOCOL

In order to present the general theoretical model, this is an example of a KERBEROS protocol with several KKDS apportioning Client and Server Keys.

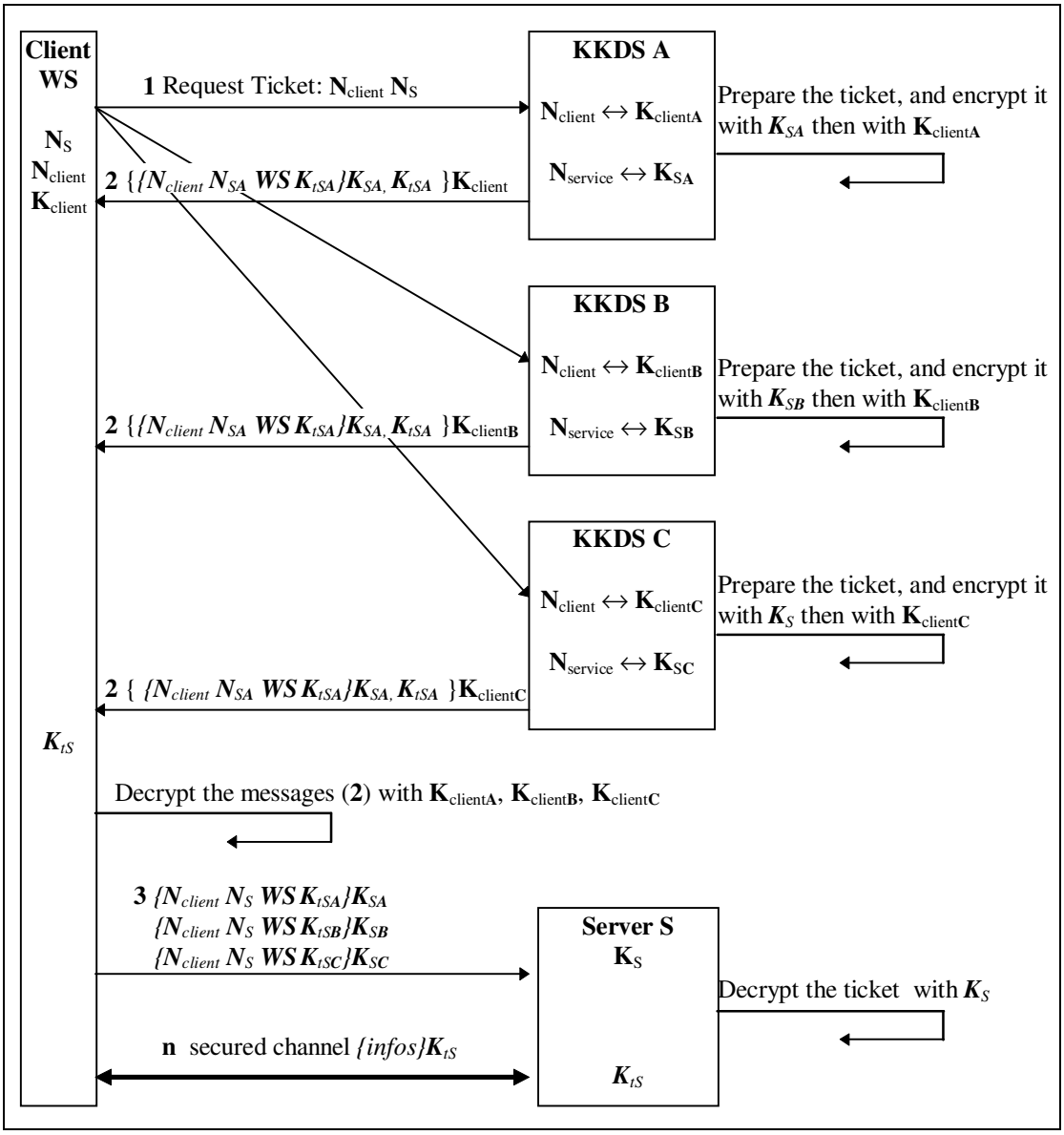


Figure 2 Sample of Key apportion applied to KERBEROS

1) The client requests a Ticket for a service S to the KKDS Set(Here 3 servers: A,B,C). The Work station knows its name and the name of the service. The message is forwarded in cleartext on the network to all KKDS servers.

2) Each KKDS looks up N_{client} and N_S, finding the private keys $K_{clientI}$ and K_{SI}. They prepare their Tickets: $N_{dient}\ N_S\ WS K_{tSI}$, enciphered them with K_{SI}. Where K_{tSI} is a temporary key$_I$ used for the secure channel. Then they send the following message enciphered with $K_{clientI}$ to the client:

$$\{\{N_{dient}\ N_S\ WS K_{tSI}\}K_{SI}\ K_{tSI}\}K_{clientI}$$

3) The Work Station decrypts incoming messages using the $\mathbf{K}_{\text{clientI}}$ password. Then send the tickets $\{\boldsymbol{N}_{client}\ \boldsymbol{N}_S\ \boldsymbol{WSK}_{iSI}\}\boldsymbol{K}_{SI}$ to the server S and store $\boldsymbol{K}_{iSA}$, $\boldsymbol{K}_{iSB}$, $\boldsymbol{K}_{iSC}$ for further communication with S. If one of the messages is not encrypted correctly with $\mathbf{K}_{\text{clientI}}$, It is just considered as "faulty".

4) The Server S decrypts the tickets using its Key $\boldsymbol{K}_{SA}$, $\boldsymbol{K}_{SB}$, $\boldsymbol{K}_{SC}$, and store $\boldsymbol{K}_{iSI}$ for further communication with the Work Station. The first communication can be to exchange a single real-session Key by encrypting it 3 times: $\{\{\{\text{Ksession}\}\boldsymbol{K}_{iSA}\}, \boldsymbol{K}_{iSB}\}\boldsymbol{K}_{iSC}$. If one of the messages is not encrypted correctly with $\mathbf{K}_{SA}$, It is just considered as "faulty". The application can define how many "valid" messages are required to grant the access.

A complete session is describe in (Table 3 p 12).

3.2.5 GENERALIZATION

General Architecture:

KKDS(KERBEROS Key Distribution Server)

In order to protect KKDS against intrusion its keys can be apportioned on "n" KKDS' servers. No specific modifications are required on the KKDS implementation. However, the key distribution procedures must be modified to allow keys to be apportioned and distributed on several servers.

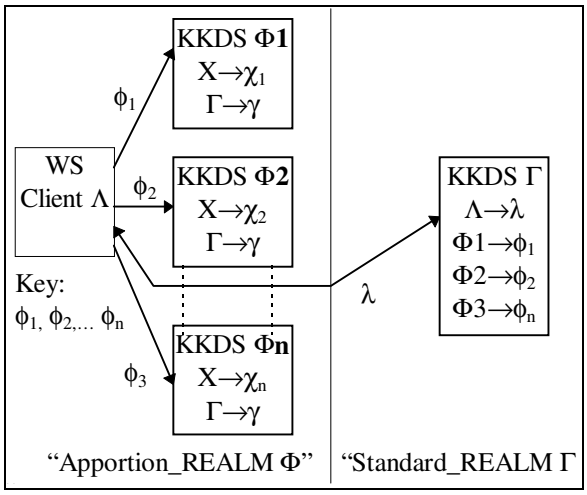


Figure 3 - Compatibility with Standard KERBEROS Independent Realms

The inter-realms' communications don't need to be modified. And the compatibility with standard KERBEROS versions can be implemented with few modifications by simulating a complex key in several instances of a single key (like the triple-DES principle). In the Figure 3 , the client **X**(realm **Φ**) has the key $<\chi_1\chi_2\ldots\chi_n>$. The Client **Λ** is not in the "Realm**Φ**" but from a "Realm **Γ**". If it wants to access a **Φ**-server. It first need to be identified by the **Γ**-server. Then requiring access to the **Φ**-clients. And last, requesting to open an application session from the **Φ**-servers. However in such a situation the security level of the "composed" protocol is the same as the lowest protocol involved in the exchange(standard KERBEROS).

Work Station KERBEROS Client

Few modifications are required on the Works Station KERBEROS Client. The WS must send its request to n servers instead of just one. It must be able to split the user password into key pieces (64bits=8 characters). Another alternative(virtual smartcards) is presented in "0

Key Management p11". It must also wait for several KKDS messages[3] in order to send its request to the application server. And last point, to maintain KERBEROS performances[4], it must start its application session by

[3] In a Fault Tolerant scope, the request must be send after a specific delay to allow to process despite unavailable servers.

[4] Cf. 3.3.2 Performances, p 9

getting a new standard key(not apportioned). For this last transaction the message must be encrypted between p and n times depending of the number of "trusted"[5] answers.

Application Server - KERBEROS client

The application server will receive several messages. According to its own policy, it will grant or not the access. This approach (server-specific policy) allows a large flexibility to design "more or less" secure policy depending of the asset to protect. It can also be used to define some Multi-Level, Multi-Group policies. For example, a server can have an access table defining access as follows:

1) Allows access if a majority of servers from the "Aristote_base" group are trusted.

2) Allows access for any trusted server from the "Aristote_secure" group(fully fault Tolerant, but assuming this servers as fully secure)

3) Allows access if all "Mephisto_base" servers are trusted

KKDS'	REQUISITE
group Aristote_base	5/8
group Aristote_secure	1/3
group Mephisto_base	3/3

Table 1 Access pre-requisite

The first access policy is the standard one, the second allows a "fully fault tolerant" system, the third one can be used as a full accreditation policy. The KKDS(in fact here, I think about the TGS) servers can be modified to grant keys only under certain conditions: specific time, the entry of a password by a specific cleared person… For example: in a Bank, a modification by an employee of the amount of money of its own account can require the countersign of the Financial Director and the Bank Manager.

Servers and "Granting" Users can be treated in the same way. Moreover, a rule language similar to router tables can be implemented.

Note: If a server has been "perverted" and "mimicked" there is no way to discover it. Hopefully, the number of perverted servers must be >(n-p), to allow an intruder to use Denial of Service. If this number go beyond p, the intruder can Disclose and modify information.

[5] a "trusted" answer is not necessarily trustworthy! A server can be "mimicked" by an intruder.

3.3 MODEL EVALUATION

3.3.1 EFFECTIVE KEY LENGTH

The apportion key enhancement requires to split the key **K** in several pieces. At a first glance we can expect the size of theses pieces to be $|\mathbf{K}|/\mathbf{n}$. However, to allow "intrusion tolerance", while conserving an acceptable effective key length, the length of key pieces must be calculated on the basis of the Policy applied.

For a majority policy: The size of the Key pieces must be so that if an intruder perverts all-but-one required server the last key piece must be $|\mathbf{K}|$.

For a policy requiring p trusted servers out of n. the size of the key pieces to prevent disclosure (for denial of service the majority case apply) must: be:

$$|\mathbf{K_i}| = |\mathbf{K}| - (2p-n)+1$$

i.e.: The intruder has to get (2p-n) keys, so we can reduce the size of each key by (2p-n). For example:

- $|K|=8bits \leftrightarrow 2\ keys\ of\ |K|=7bits,$
- $|K|=8bits \leftrightarrow 4\ keys\ of\ |K|=6bits,$
- $|K|=8bits \leftrightarrow 128\ keys\ of\ |K|=1bit$

Practically, the most interesting implementation is probably to keep a fixed length as in the "majority" policy since no dynamic cryptographic algorithms are required.

Another point of interest about Key length is the effective key length of the message (4) $\{\ldots.\{\{\{Ksession\}\boldsymbol{K_{S1}}\}, \boldsymbol{K_{S2}}\}\ldots.\}\boldsymbol{K_{SN}}$ composed of **N** successive encryption of the same message. Studies of such cipher composition are far beyond the scope of this paper, however one must be aware that for instance the effective 3DES[6] key length is only around 112bits long. This is especially true if an intruder knows(and has chosen) p keys. However, in a real time implementation, even if the resulting key effectiveness is lowered, the life-time of each key can be set to disable any brute-force attack on these weakened keys.

[6] *"Triple DES with independent keys is not, as naively might be expected, as difficult to break by brute force as a cryptosystem with three times the key length. A space/time tradeoff has been shown which can brute-force break triple block encryption in the time naively expected for double encryption." (RFC1851)*

3.3.2 PERFORMANCES

The use of "Apportion" keys increase considerably the number of messages required to enable a session. For a n-servers configuration:

messages type	number of messages	enc/decryption
(1) request tickets: from client to server	n	n
(2) tickets send from servers to client	n	n
(3) tickets send to server	1 long or n shorts	n
(4) establishing "permanent key"	1	2n

Total: (**3n+1** short messages) or (**(2n+1)** short messages **+ 1** long one)

Instead of **3** messages in the standard KERBEROS protocol!

However, the real overhead results in the encipherment time. This is especially true in all Fault & Intrusion Tolerant Applications. During this initial session establishment, the number of encryption/decryption required is **5n** instead of **3**. To limit this overhead, and to keep performances near a standard KERBEROS protocol, the message(4) is used to set a secure channel using a common short(fast) session key. This protocol is based only on shared secret keys, considerably faster than public key algorithms.

The required number of servers and size of data increases by a factor **n**.

3.3.3 IMPROVEMENTS

Inter-Realms architecture

By distributing, the architecture of the KKDS, a classical problem of distributed system occurs: How to distribute the trust through "globally trusted" set? Once again the following schemes are greatly simplified, however they present the main ideas.

The mechanism to authenticate users across realms is provided by the fact that each KERBEROS server is also a client for other realms KERBEROS servers. This mutual client relationship between the KERBEROS services allows a client to authenticate itself to the KERBEROS server of a foreign realm. The KKDS must also shared an inter-realms key.

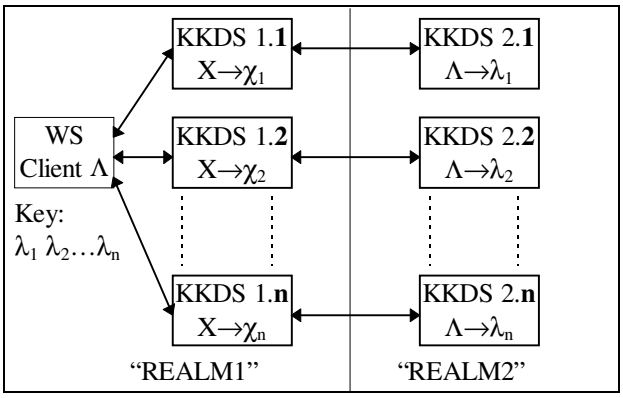


Figure 4 a "one-to-one" Architecture

"Intersection" Architecture

This Architecture doesn't requires any modifications into the KKDS code, optimize the number of messages, but greatly weaken the Fault Tolerance of the system.

The client identity is established by each pair of KKDS of same level but from two separate realms. However if only one of the two KKDS is perverted by an intruder or out of order, the whole pair is unusable. If such chaining is really long, the risk increases…

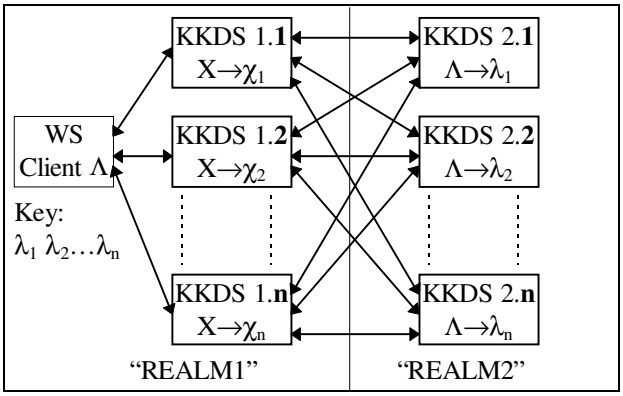
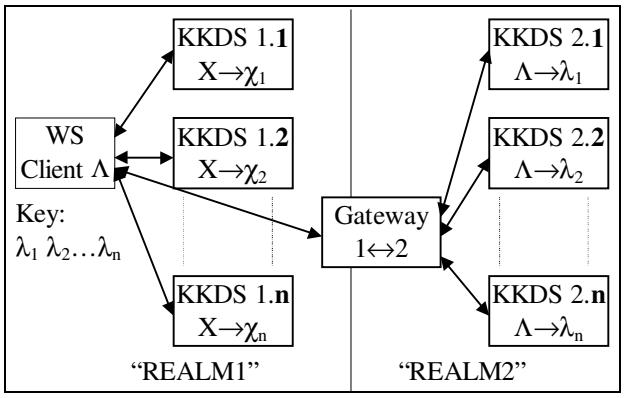


Figure 5 fully multicasted Architecture

"Association" Architecture

This kind of Architecture requires some modifications into the KKDS code, increases the number of messages by $n_1 * n_2$, and greatly weaken the Intrusion Tolerance of the system, but maintain a high Fault Tolerance.

Each KKDS, has all the required keys to gain full access of the user. If an intruder perverts any of them he can take the control of the client session.



"Hybrid" Architecture

Here is my proposal, it doesn't require any major modification in the KKDS, it optimize the number of messages, and keep the same level of Fault & Intrusion Tolerance all over the realms.

The gateway is nothing more than a host able to forward a request between several realms in the same way than an IP gateway. To be really effective, the gateway (and the client) should be able to not sent a request to a set of servers if they are not in the same realm. It should have a default "route" and "realms routes". There is no sensible information on the gateways. They don't require any encryption features. However, to assure Fault Tolerance there should have backup Gateways. In a real environment, we can assume that the Realms won't change often and a "static" routing algorithm will be powerful enough for such application.

The "Gateway" can also be placed on the same host than the KKDS, in such situation the *"figure 5"* scheme apply. However keys won't be deciphered on the server and so the Intrusion tolerance principle is respected. A problem occurs when a user from one realm wants to access to an application from another realm. The only way to deal with this problem is to apply a mix between the *"schemes 4 and 5"*. In such a case, several gateways can be used. The client, opens a session with the gateways of its realm then each gateways send requests to the KKDS of the application realm, etc…

Key Management: virtual smartcard

One of the major drawbacks to use fragmented keys is that their size can't really be reduced. As a result if the "key array" is too large no human user would be able to remember a too long password[xi]. In his Secure distributed system[ii], **Mr. Deswarte** points out the difficulty to manage such keys and uses smartcards.

In this architecture (but it can be extend to many other), the user can keep a relatively small password and use a large key. If each server stores an encrypted image of the user keys (a virtual smartcard), they just have to send this image at any opening session request. Then, in order to decrypt its session keys, the user will have to decrypt its smartcard with its password. Because the "smartcard " is located on several servers, there is "fault tolerance". Moreover, the smartcards are encrypted on each server so an Intruder can't use them. Another advantage of such a "virtual smartcard" is that any new server, application or client declaration can be done directly on the servers' level by inserting a new field. There must be a tradeoff between "management ease" and security. All the involved server administrators (or at least p of them) must cooperate, by typing their passwords to allow such modifications on the set of servers.

Entity	**Client δ**	**Server 1**	**Server 2**	**Server 3**
Knows **request smartcard**	password $K_{\delta}0$ •req. sc → S1 •get $K_{\delta}1$, $K_{\delta}2$, $K_{\delta}3$ from $\{K_{\delta}123\}(K_{\delta}0)$ by decrypt. with $K_{\delta}0$	$K_{\delta}1$, $\{K_{\delta}123\}(K_{\delta}0)$ •receive request •send $\{K_{\delta}123\}(K_{\delta}0)$	$K_{\delta}2$, $\{K_{\delta}123\}(K_{\delta}0)$	$K_{\delta}3$, $\{K_{\delta}123\}(K_{\delta}0)$

Table 2 smartcard acquisition

To avoid an intruder to "fake" smartcards, they must be composed of the keys, of a signature (SIG) and a checksum (CRC). Moreover, to avoid passive sniffer attacks, the message should be encrypted. The fact that a smartcard can be partially composed of a key local to a server can weaken some cryptographic algorithms. In such a case the smartcard located on the server 2 can have the following form: $\{\{K_{\delta}1\}(K_{\delta}0), \{K_{\delta}3\}(K_{\delta}0), \{CRC,SIG\}(K_{\delta}0)\}$.

The system must also tolerate uncompleted sessions to deal with some attempt to spoof the network with old messages. The KERBEROS protocol uses time-stamps and time-out to limit such attacks. The network and transport protocols must also be checked to avoid any inconsistency (desynchronized state) in their model.

One of the most delicate problems with "real" smartcard is the way to update them. The table below gives an example of a user update to modify its password. He will first have to access the "change password" application by establishing a "secure" session. Then "change password" must validate its new password (it can looks up into a dictionary, restrict the size,…) and prepare a changing ticket(to avoid a user to change his password by bypassing this control). The new password is used to encrypt the smartcards on the client. The newly generated $\{smartcards\}(K_{\delta}0N)$ are then sent with the "change ticket" encrypted with the old client Key portions corresponding to each server.

Entity	Client δ	Server 1	Server 2	Server 3	Application
Knows	password $K_{\delta 0}$	$K_{\delta 1}$, $\{K_{\delta 123}\}(K_{\delta 0})$ K_{cp1}	$K_{\delta 2}$, $\{K_{\delta 123}\}(K_{\delta 0})$ K_{cp2}	$K_{\delta 3}$, $\{K_{\delta 123}\}(K_{\delta 0})$ K_{cp3}	K_{cp1}, K_{cp2}, K_{cp3}
request smartcard	•req. sc → Server 1	•receive request			
•get smartcard	•get $K_{\delta 1}$, $K_{\delta 2}$, $K_{\delta 3}$ from $\{K_{\delta 123}\}(K_{\delta 0})$ by dec. with $K_{\delta 0}$	•send $\{K_{\delta 123}\}(K_{\delta 0})$			
•request to change passwrd	•req. "chpasswd"				
•session Keys (Ticket: $\{K_{si}\}K_{cpi}$) generation.		•receive req. •generate K_{s1} •send $\{\{K_{s1}\}K_{cp1}, K_{s1}\}K_{\delta 1}$	•receive req. •generate K_{s2} •send $\{\{K_{s2}\}K_{cp2}, K_{s2}\}K_{\delta 1}$	•receive req. •generate K_{s3} •send $\{\{K_{s3}\}K_{cp3}, K_{s3}\}K_{\delta 3}$	
• get session keys and tickets	• get K_{S1}, K_{S2}, K_{S3} from the messages by dec. with $K_{\delta 1}$, $K_{\delta 2}$, $K_{\delta 3}$.				
•send ticket to aplication	•get tickets •send $\{K_{s1}\}K_{cp1}$ $\{K_{s2}\}K_{cp2}$ $\{K_{s3}\}K_{cp3}$				
• get session keys from tickets					•receive tickets •get K_{S1}, K_{S2}, K_{S3}
• generate and send effective session key					•generate Ks • send $\{\{\{K_s\}K_{s1}\}K_{s2}\}K_{s3}$
• get effective session keys from tickets	•get K_S from the messages by dec. with K_{s1}, K_{s2}, K_{s3}. •start transactions				
	• request a new password K_{0N}				• establish the new password **K_{0N}** • a changing ticket for each server encrypted with K_{si}
•generate the new smartcard	• encrypt $K_{\delta 1}$, $K_{\delta 2}$, $K_{\delta 3}$ with K_{0N} to generate the new smartcard, add CRC, signature, eventually split it into pieces. • send it back to the servers, encrypted with the old Key, with a signature and a CRC				
• servers update		•check CRC, ticket •check signature • replace $\{K_{\delta 123}\}(K_{\delta 0})$ with $\{K_{\delta 123}\}(K_{\delta 0N})$	•check CRC, •check signature •replace $\{K_{\delta 123}\}(K_{\delta 0})$ with $\{K_{\delta 123}\}(K_{\delta 0N})$	•check CRC, ticket •check signature • replace $\{K_{\delta 123}\}(K_{\delta 0})$ with $\{K_{\delta 123}\}(K_{\delta 0N})$	

Table 3 Complete Transaction to change a user password, the session opening is surrounded by double lines

The same protocol can be used to manipulate servers or application keys. However, this protocol isn't fully consistent because if an intruder can have access to the "change password" application, he can grab newly generated keys. The only solution that I found is to migrate the "changing password" application from a "non-necessarily trustworthy Host" to the client. The password generation can then be done on the client works-station. This "code migration" requires some classical precautions (checksum & signature), but also that the client works-station is clean of any intruder. In fact the risk is just moved from a "public" host to a supposedly "private" one. Another alternative could be to "apportion" the "change password" application. However, the password checking requires the full password to process[7].

> *This "special" session can also be used to centralize a backup of keys of the system. Centralizing all keys is exactly what this protocol attempt to avoid, however if a list of the keys is required and can be securely stored… They can be collected at this protocol point.*

3.4 CONCLUSION

The "key apportion model" presented in this paper doesn't match exactly the puzzle idea. In fact, fuzzy logic could be an interesting way to represent and deal with "*fuzzy*" keys. This principle can also be adapted to the applications and to the client. When applied to the client it allows interesting multi-accreditation schemes. It can be reinforced with a more complete protocol: FRD (Fragmentation Dissemination and Replication)[iii,iv,v].

The "virtual" smartcard notion is an interesting tradeoff to avoid hardware authentication while allowing long shared keys. Another solution to allow virtual smartcard "ease of maintenance" and hardware would be to store a long "K0" Key on a "real" smartcard.

The modifications required in the KERBEROS protocol are essentially at the level of the client and application. Such kind of architecture should greatly reduce the risks of bribery by requiring several administrators. It also improves Fault and Intrusion tolerance in a heterogeneous environment. (considering "security holes" specifics to some Operating Systems or applications[xii]). However, KERBEROS will keep its weakness[xiii].

4. ACKNOWLEDGEMENTS

This small enhancement is based on a "*Special Topic in Computer Security*" course proposed by **E. Amoroso**(ATT-STEVENS). The preparation of this paper has been greatly assisted by E-mail discussions with **G. Eizenberg**(CERT), **E. Baize**(BULL) and especially with **Y. Deswarte**(LAAS).

[7] This idea, to execute the "almost securely" the application on the client Workstation, is directly inspired by the file system developed by the LAAS[iii].

5. REFERENCES

[i] **B. Cheswick**, "The design of Internet gateway", AT&T Bell Laboratories
An example of Gateway building, and general security consistency.

[ii] **Michael K. Reiter**, "Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart", Proc. Of the 2nd ACM Conference on Computer and Communication Security, nov. 1994,pp.68-80.
The Rampart Toolkit allows to build High-Integrity Services by using voting between duplicated servers.

[iii] **Yves Deswarte, Laurent Blain, Jean-Charles Fabre**, "Intrusion Tolerance in Distributed Computing System", Proc. Of the IEEE Symposium on research in Security and Privacy, Oakland (USA), may 1991, pp110-121.
This paper describes how to use distributed architecture properties to build Intrusion Tolerant systems. An example of secure distributed file system is presented.

[iv] **Jean-Charles Fabre, Yvex Deswarte, Laurent Blain**, "Tolerance aux fautes et securite par fragmentation, redondance, dissemination", TSI, volume 15,numero 4, 1996, pp. 405-427.
I could'nt found it until know, however it seems to be the most recent paper about TRD architectures.

[v] **Brian Randell, Jean-Charles Fabre**, "Fault and Intrusion Tolerance in Object-Oriented Systems",0-8186-2265-2/91 $1.00, IEEE, 1991, pp.180-184
A very interesting way to use OO properties on the same model than distributed architecture to protect and disseminate Data.

[vi] **S.P. Miller, B.C. Neuman, J. I. Schiller and J.H. Saltzer**, "Kerberos Authentication and Authorization System", Project Athena Technical Plan, section E.2.1, MIT, Oct. 1988.
Certainly the best paper to understand the Kerberos protocol.

[vii] **J. T. Kohl, T. Ts'o, B. C. Neuman**, "The evolution of the Kerberos Authentication Service", paper presented at the Spring 1991 EurOpen Conf.
This paper presents the differences between versions 4 and 5 of Kerberos.

[viii] "Kerberos V5 Implementer's Guide", MIT Information Systems, June 1996
T. Ts'o, "Notes on Building and Installing Kerberos V5, Edition BETA 6", Mar, 1996
Two papers about Kerberos, required before installing and hacking into it.

[ix] **M. Bellovin, Merritt** "Limitations of the Kerberos Authentication System", AT&T Bell Laboratories, Computer communications review, Oct 1990
A famous paper presenting several limitations of Kerberos.

[x] **G. Perec**, "La vie mode d'emploi", Hachette-litterature 1978

[xi] **W. Belgers** "UNIX Password Security", Dec. 1993
A little paper to remember without efficient users password, the most complex Security Architecture is useless…

[xii] **D. K. Jess, D. R. Stafford, U. W Pooch**, "A UNIX Network Protocol Study: Network Information Service" , Texas A&M University.
A good example of unsecure application.

[xiii] **L. Joncheray**, "A simple Active Attack Against TCP", Merrit Network inc., Apr 1995
An example of attack efficient against Kerberos (without secure TCP). Less simple with key apportion it is still quite efficient.