Prototyping an Armored Data Vault
Rights Management on Big Brother's Computer [⋆]

Alex Iliev and Sean Smith

Department of Computer Science/Institute for Security Technology Studies
Dartmouth College
{sasho,sws}@cs.dartmouth.edu

DRAFT of April 22, 2002

Abstract. This paper reports our experimental work in using commercial secure coprocessors to control access to private data. In our initial project, we look at archived network traffic. We seek to protect the privacy rights of a large population of data producers by restricting computation on a central authority's machine. The coprocessor approach provides more flexibility and assurance in specifying and enforcing access policy than purely cryptographic schemes. This work extends to other application domains, such as distributing and sharing academic research data.

1 Introduction

This paper presents a snapshot of an ongoing experimental project to use high-end secure coprocessors to arbitrate access to private data.

Our work was initially inspired by Michigan's *Packet Vault* project [2], which examined the engineering question of how a central authority, such as law enforcement or university administration, might archive traffic on a local area network, for later forensic use.

We perceived in addition to the engineering question a Digital Rights Management (DRM) aspect of an unusual kind. In the standard DRM scenario, a large data producer seeks to control use of its data by multiple "small" users. We see the possibility to turn this around and enable individual users to control with confidence how their data is to be used by a powerful authority ("Big Brother"). Why would people agree to have data collected at all? For network traffic in particular, some benefits of collection are certainly plausible, such as:

– the ability of administrators to diagnose whether a particular network attack occurred;
– the ability of law enforcement to gather evidence for illegal activity that the community regards as sufficiently egregious;
– the ability of scientists to analyze suitably sanitized Web surfing activity of consenting individuals.

[⋆] Published at *Privacy Enhancing Technologies 2002*. © Springer-Verlag.

How do we restrict the usage rights on archived data to exactly socially prescribed policy—especially given human nature's inclination to exceed authority?

We designed and prototyped a system using commercially available high-assurance secure coprocessors. We felt that a working prototype would further several goals:

– validating that this approach to user privacy works in practice;
– demonstrating the advantages of a *computational* approach to rights enforcement, compared to strictly a cryptographic one;
– demonstrating the application potential of current commercial off-the-shelf (COTS) secure coprocessor technology;
– demonstrating a socially equitable use of tamper-respondent technology: rather than impinging citizens' computation, it is restricting Big Brother on their behalf.

We believe this work will have applicability in other domains of protecting community rights—such as key escrow and recovery in a university PKI. We discuss this further in Sect. 6.3.

This Paper. Section 2 describes in some detail the motivations behind the Packet Vault and our Armored Vault. Section 3 describes our design, and Sect. 4 our implementation of the prototype. Section 5 has our discussion of the prototype implementation. Section 6 discusses the wider applicability of our design, and what we plan to do next.

Code. Our implementation is available for public download[1], as is the developer's toolkit[2] for the IBM 4758 platform.

2 Background

In this section we examine some developments motivating our prototype's subject matter of comprehensive network traffic archival. Then we describe the Armored Vault—the original archival tool on which we base our prototype, and the reasons for seeking to armor such a vault. Finally we list some other works relevant to our topic.

2.1 Evidence collection

Government authorities on both sides of the Atlantic are keen to get their hands on network traffic. In the US, this is done by the FBI with the Carnivore tool. Carnivore is a software system designed to run at an ISP and collect communications of the parties under surveillance. [9, 18, 3] As part of provisions for combating terrorism, European Union governments are seeking to revise the

[1] `http://www.cs.dartmouth.edu/~pkilab/code/vault.tar.gz`
[2] `http://www.alphaworks.ibm.com/tech/4758toolkit`

EU Directive on data protection and privacy of 1997[3] to allow for retention of telecommunications data in cases of national security significance, apparently without requirements for case-by-case court authorization and for minimal targeting of specific suspects. [23] The Carnivore looks tame in comparison to this beast.

2.2 Complete Archival of Net Traffic

2.3 The Packet Vault

Storing all network traffic is on the agenda, and warrants some attention. In addition to increased surveillance power and convenience for law enforcement, traffic could be stored for network intrusion evidence, or authorized research.

The Packet Vault is a pioneering device to address the plentiful security questions posed by a complete record of all network traffic [2]. The follow-on Advanced Packet Vault can keep up with a 100 Mbit ethernet [1]. A design goal is to store packets securely, so that they may be accessed only through the security mechanism imposed by the vault: in an archive (on a CD-ROM) host-to-host *conversations* are encrypted with separate secret keys, and the set of these keys is encrypted using a public key algorithm with the private key held by a trusted entity, the vault *owner*. Note that this owner is a person or persons—the Regents of Michigan University were the owners of the first prototype Vault. Access to the archived data is accomplished by getting the owner to decrypt the secret keys for the desired conversations, which they presumably do once they decide all access conditions are satisfied.

Weaknesses. The Packet Vault approach to securing archives leaves some important security questions:

Vulnerability to insider attack. The Vault depends on the trustworthiness of the vault owners. If they are law-enforcement personnel for example, trusting a complete record of network traffic to them may be objectionable to many users. It is not very different from letting them have a clear-text record of the net traffic, and trusting them to use it as all concerned parties (whose data is stored stored) would like. Even if the owners act in good faith, their private key may be compromised, which could either expose all the archives on which the public key of the pair was used, or necessitate those archives' destruction.

Access flexibility. Accessing the archived packets is not very flexible. Full access or no access to some set of conversations can be granted to a requester; nothing else is possible. Some other useful access possibilities include:

– *Accessing data at finer granularity than a conversation.* Rounding to the nearest conversation could either skip needed data, or produce excess data to the extent of making the released data unsuitable for evidentiary use. [4]

[3] Directive 95/46/EC

[4] Wiretap authorizations often include a minimization requirement—data collected is to be strictly limited to that needed for the investigation

Access to computation seems to provide more power in selecting packets than a purely cryptographic scheme would.

– *Postprocessing before output.* Such a capability could be useful in a lot of applications—deciding the presence or absence of something in the archive without revealing additional details, anoymizing data for experiments (eg. blanked source headers), or producing statistics of the net traffic.

  It is not clear how a some simple cryptographic extension to the Packet Vault could provide this particular feature. By definition, computation must be done on packets which must not themselves be seen in their original form. Proving that *filtered* data correctly follows from genuine archive data, without disclosing the archive data, appears difficult to solve *in general* using cryptography alone.

2.4 Related Previous Work

These are some works related to policy expression and compliance checking, the use of secure coprocessors for controlling access to data, and distribution of sensitive data with attached access policy.

An early comprehensive and general-purpose system for policy specification and checking was *PolicyMaker* [5, 6]. It has since evolved into *KeyNote* [4]. The SPKI system [10] deals with authorization in a distributed environment. The *Trust Policy Language (TPL)* [11] is another suggested approach to policy expression and checking. A system such as these would be appropriate to plug into a more complete version of our prototype to provide a usable mechanism to specify and check access policy.

Policy-carrying, Policy-enforcing digital objects apply enforcement of computational policy to data. They use language-based control (inlined reference monitors for Java) to enforce compliance. [15]

A use of secure coprocessors in controlling access to data is described in [25]. It covers protection of executable code, by encrypting the code such that only the designated coprocessor can access it. This falls more in the traditional DRM scenario of protecting data of a large organization against individual users. There is no discussion of access controls finer than "only coprocessor X can run this code". Also the coprocessors used[5] were hand-built and not commercially available.

Our work provides an interesting counterpoint to conclusions that controlling access to data by a large population is inherently doomed [24]. In our scenario, a large population controls access to their data by a small group, and because the access points are few, they can be controlled by high-end devices which provide high assurance.

[5] Dyad, utilizing IBM Citadel hardware and a Mach 3.0 microkernel

3 Prototype Design

3.1 Overview

All the interesting features mentioned in Sect. 2.3 (flexible packet selection, post-processing) could be provided by trusting the vault owners to perform them, for example perform calculations on the stored packets and give us results. Trusting refers not just to the integrity of the owners, but also to the integrity of the machines on which they perform their computations.

We make use of secure hardware to be the "trusted party" in the archival system, replacing the vault owners previously described. Call this trusted party Solomon for the duration of this section. Solomon will serve us as follows:

– He possesses an encryption key-pair and a signing key-pair, a description of how to evaluate the "worthiness" of requests for data, and a description of what computation must be done to select and process traffic data to be given to a requester.
– The stored traffic is all encrypted with Solomon's encryption key. If we do not wish to trust someone to do this part, we could get Solomon to do it too.
– Requests for access to the stored data are given to Solomon, who can then (1) evaluate if the request is worthy of being honored, (2) compute what data is to be released and (3) perform whatever computation is needed on that data to produce the final result, which he then signs and releases.

Concretely, we use two secure coprocessors—one for collecting the net traffic and producing archives, and the other for arbitrating access to the archives. We refer to them as the *Encoder* and *Decoder* respectively. We use two because their tasks could be separated by space and time, and also the job of the Encoder is high-load and continuous.

The Encoder encrypts the stored traffic using the Decoder's encryption key, and signs the archive using its own signing key. The Decoder must decide if access requests against archives are authorized, decrypt the archive internally (no one outside can observe it decrypted) in order to perform selection of the desired data, compute the query result using the selected data and sign this final result. Fig. 1 shows an overview of the archive generation and access procedures.

3.2 The Secure Hardware

We use the IBM 4758 model 2 programmable secure coprocessor [17, 19] [6]. Some properties of the 4758 are:

[6] Note that the attack by Bond and Anderson [7] penetrated a version of the CCA software running on the 4758, but not the security provisions of the device itself. Those remains secure.

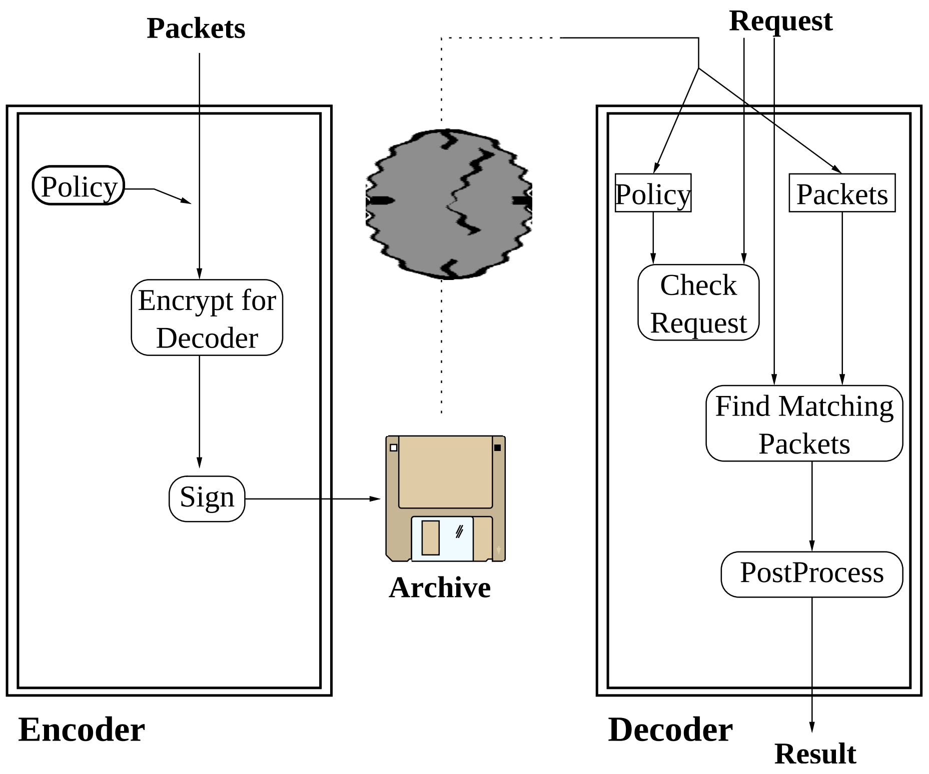

Fig. 1: Overview of armored vault design. Details of the cryptographic organization are described in Sect. 3.4.

- It can be programmed in C, with full access to an embedded Operating System (CP/Q++ from IBM) and hardware-accelerated cryptographic functionality. Programs run on a 99-MHz Intel 486 processor, with 4 MB of main memory on the 002 model.
- With high assurance, it can carry out computation without possibility of being observed or surreptitiously modified.[7]
- It can prove that some given data was produced by an uncompromised program running inside a coprocessor. This is done by signing the data in question with a private key which only the uncompromised program can know. Details of this process follow.

The coprocessors are realized as PCI cards attached to a *host* workstation. Currently drivers exist for Windows NT/2000, Linux, AIX and OS/2 hosts.

Outbound Authentication. The 4758 provides a mechanism for proving that output alleged to have come from a coprocessor application actually came from an

[7] The 4758 platform was the first device to receive FIPS 140-1 Level 4 validation [14, 21].

uncompromised instance of that application [19]. This is achieved by signing the output with an *Application key-pair*, generated for the application by the coprocessor, and providing a certificate containing the public key of the signing pair, and a certificate chain attesting to this certificate. In this chain is the identity of the application too. The structure of the chain is shown in Fig. 2. To establish that the coprocessor application which produced some signed output is the expected one and is uncompromised one would:

1. Verify the signature using the public key in the application certificate.
2. Verify that all the certificates are signed as appropriate by their parent certificates.
3. Verify that the identity of the application, stored in the Layer 2 certificate, is what we expected. This identity includes things like program name, developer ID and program hash. The identity of the OS in Layer 2 can also be verified.

When this is done, we can be certain that the correct program produced the signed output.

3.3 Access Policy

We gather all the information relating to how archives are to be accessed into an *access policy*. The policy is thus the central piece of the armored vault. The Decoder will give access to the archive only in accordance with the access policy, and *no one* can extract any more information, since the design of the coprocessor precludes circumventing its programmed behavior.

To represent access policies we use a table, whose rows represent different *entry points* into the data, and whose columns represent the parameters of each entry point. Anyone wanting access must select which entry point to use, and then satisfy the requirements associated with it. The parameters associated with entry points are:

Request template A template of a query selecting the desired data, with parameters to be filled by a particular access request. We call the format of the query the *data selection language*. An example could be "All email to/from email address X".

The following two parameters define how the decoder decides whether a request is legal.

Macro limits Limits on various properties of the result of the query. These could be things like total number of packets or bytes in the result, how many hosts are involved in the matching data, or packet rate with respect to time at the time of archival.

Authorization The authorization requirements for this entry point. These could be something like "Request must be signed by two district judges".

This parameter defines how the Decoder will compute a final result for the query based on the initial data selected.

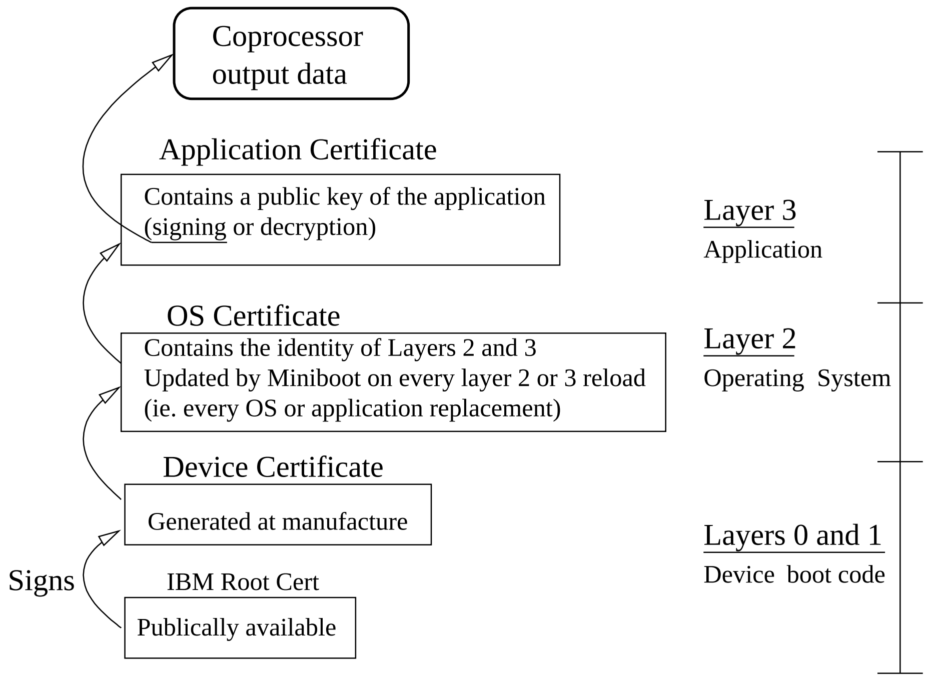

Fig. 2: Simplified certificate chain attesting to an application key-pair which signs output data. The OS certificate contains the application's identity information. *Layers* in the 4758 represent progressive stages in the startup process, as well as a progressively tighter security environment, essentially with higher layers unable to observe or modify data in lower layers.

Post-processing A filter to apply to the data picked out by the query. Some examples could be "Scrub all IP addresses" or a statistical analysis of the data.

The procedure for requesting data from the archive will be to indicate an entry point, and provide parameters for its request template. The request will contain the authorization data needed to satisfy the requirements of the chosen entry point, for example be signed by all the parties who need to authorize the access.

The actual policy has to balance the opposing motivations of (1) enabling all acceptable queries[8], as decided at the time of archival, to be satisfied and (2) ensuring that there is no way for anyone, even rogue insiders, to gain access to more of the data than was intended.

[8] Queries could be far in the future.

3.4 Cryptographic organization

The top-level cryptographic organization of the armored vault is as shown in Fig. 1. Here we describe the details of each step.

The Encoder must be initialized by giving it the public encryption key of the Decoder which will control access to the archives produced by this Encoder. The key is contained in an Application Certificate, part of a chain as described in Sect. 3.2. Note that the encoder can determine if the alleged decoder is genuine from the application identity in the cert. chain.

The Encoder produces an archive which is structured as shown in Fig. 3. Encryption of the stored packets is two-level—first the packets are encrypted with a TDES session key, and then the session key is encrypted with the encryption key of the Decoder. The Encoder provides verification for the archive by signing it and attaching a certificate attesting to the signature, as described in Sect. 3.2.

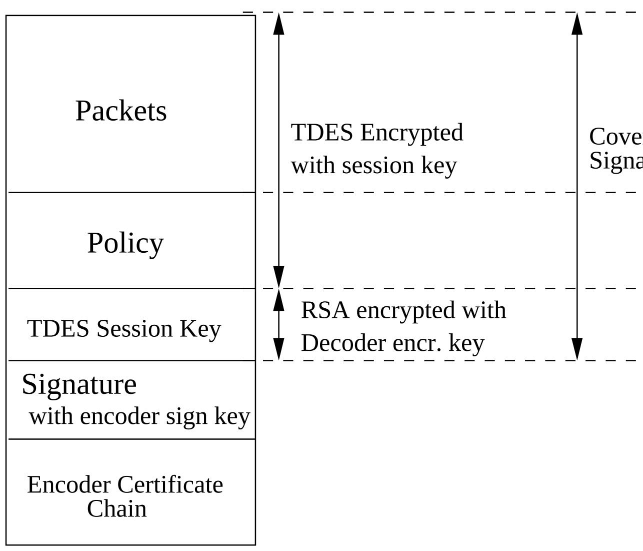

Fig. 3: Structure of the archive produced by the encoder.

When the Decoder receives a request against an archive, it verifies the archive by checking the signature and the identity and supporting chain of the Encoder which produced the archive. It then decrypts the session key using its private decryption key, decrypts the packet dump using the session key, and carries on with processing the request. When the final result is computed, it signs it in the same way that the Encoder signs an archive.

4 Implementation

4.1 Overview

Our setup consists of a Linux PC acting as the host to both the Encoder and Decoder cards. We picked Linux because we prefer open systems, and the Linux driver for the 4758 is open-source.

The user interface to the prototype vault consists of a command-line program running on the host, which performs two tasks with the Decoder:

– Request its public encryption key/certificate together with a certificate chain (see Sect. 3.2) attesting to the key, and save these in a file. For now the certificates are in the format used by the secure cryptographic coprocessor (SCC) interface of the 4758 [12].
– Run a request for data against an archive previously made by the Encoder.

and two tasks with the Encoder:

– Set the encryption key and supporting certificate chain of the Decoder this Encoder is to work with, using a file generated by the Decoder above.
– Produce an archive given a `libpcap`-format packet dump. This can only be done after a partner Decoder is established by supplying its public encryption key.

Packet dumps can be produced by any packet sniffer program which uses libpcap as its packet-capture mechanism and allows binary packet dumps to be produced. Two possibilities are the quintessential `tcpdump`, and `snort`.

4.2 Encoder operation

The Encoder takes a `libpcap`-format packet dump and produces an archive structured as shown in Fig. 3. It stores the access policy internally, and attaches it to every archive. It performs everything described in our design, but being an early prototype is limited to processing only as much data as will fit into the coprocessor at once (about 1.7 MB with our current prototype).

4.3 The data selection language

We use an existing package, `Snort` [16], version 1.7, to provide the packet selection capability in our demo. Snort is a `libpcap`-based Network Intrusion-Detection System (NIDS) which can select packets using the Berkeley Packet Filter (BPF) language as well as its own rule system which features selection by packet content as well as by header fields. This rule language is our data selection language. The snort rule system is described in detail at `http://www.snort.org/writing_snort_rules.htm`.

We chose Snort as it is an Open Source tool in active development, and active use. Important features are IP defragmentation, the capability to select packets by content, and a developing TCP stream re-assembly capability.

Porting Snort. We had to compile a subset of Snort (essentially the packet detection system) to run inside the Decoder card and interpret requests for data.

We had to supply implementations for non-STDC functions which were still needed for packet detection and processing, like `inet_ntoa` and `getprotobynumber`.

The major challenge was implementing the `stdio` functions to enable the transfer of data to and from Snort when it ran inside the Decoder. CP/Q++, the current 4758 production OS, does not provide filesystem emulation, so we wrote implementations for a few of the POSIX filesystem calls: (`open`, `write`, etc.). These functions for now operate with memory buffers inside the card.

Snort rules. Snort rules specify what packets Snort selects for further processing (like logging or alerts). They select packets based packet headers *or* packet data. A simple example to select TCP packets from the http port for logging is

```
log tcp any 80 -> any any
```

This only performs matching on packet headers. It could be read as "log TCP packets coming from any host, port 80, going to any host, any port". A fancier example using content matching to produce an alert on noticing a potential attack (whose signature is the hex bytes 90C8 C0FF FFFF) is

```
alert tcp any any -> 192.168.1.0/24 143
(content: "|90C8 C0FF FFFF|/bin/sh"; msg: "IMAP buffer overflow!";)
```

4.4 Decoder operation

The Decoder implements our design (limited to small archives which can fit in the coprocessor at once), with the following exceptions:

- No post-processing is currently possible. This will be a bit of work to implement fully, with reasonable capabilities, so we did not attack it in this prototype.
- It has no authorization capabilities, except simple macro limits.

The detailed Decoder operation is shown in Fig. 4.

4.5 Access Policy

We implement the access policy as XML-format text, with a *row* element for the entry points (rows in the policy table), inside which are XML elements for all the entry point fields, with the exception of post-processing which we have not implemented yet. We currently use very quick and simple "parsing" of the table, with an eye to use the `expat` XML parser. The table used in the current version of the prototype is shown in Fig. 5.

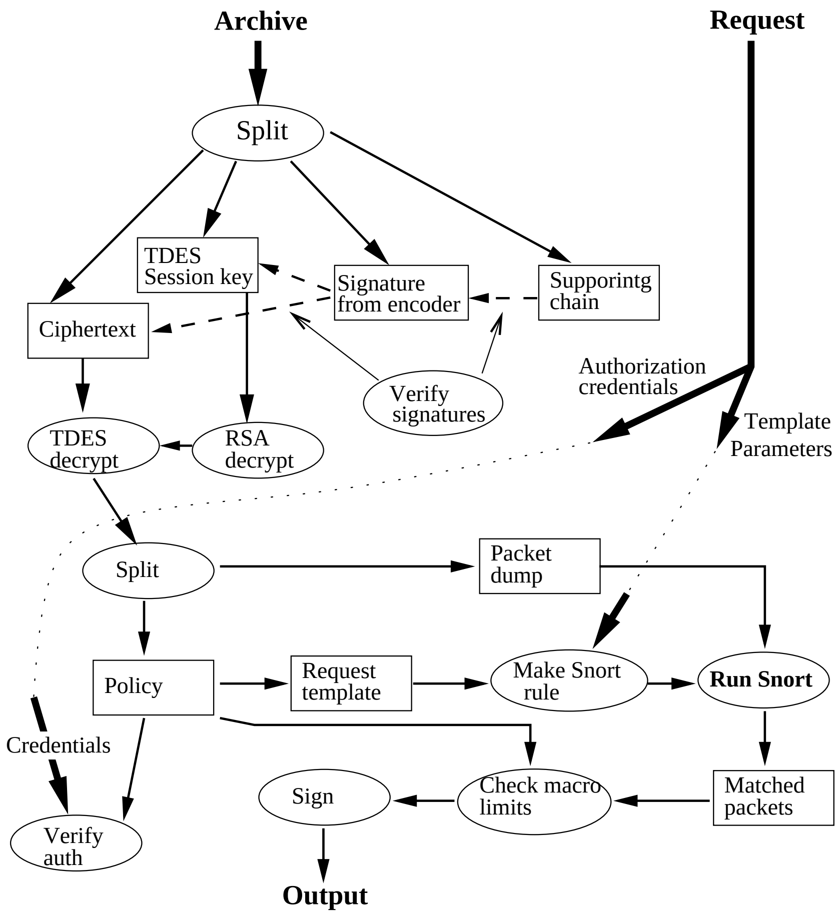

Fig. 4: Decoder operation. The Snort rule used to run Snort is made by instantiating the template in the policy using values from the request.

4.6 Request Structure

Requests consist of a set of name=value assignments, one for the row number from the policy table through which the request is going, and the rest assignments for the request template of the chosen row. An example which could be used with our sample policy in Fig. 5 is
`row=1; port=80`

5 Discussion

5.1 Our Implementation

Snort. This was one of the successful aspects of this prototype. Snort is a very capable packet detection engine, and it runs happily inside the secure coproces-

```
<?xml version="1.0"?>
<policytable>

<title>
Experimental table
</title>

<row>

<reqtemplate>
log tcp any $port -> any any (content:"sasho")
</reqtemplate>

<macrolimits>
$total_packets < 100
</macrolimits>

</row>

</policytable>
```

Fig. 5: Current prototype policy. This policy allows the selection of TCP packets containing the word "sasho", and coming from some port specified in the request (eg. "port=80"). If more than 100 packets match, the request will be declined.

sor. It will enable us to extend our policy capabilities considerably, especially when we start to consider application-level data selection, and reassembled TCP streams become important.

Policy. Our current prototype policy (in Fig. 5) is fairly basic, but it does demonstrate many key points about the armored vault:

– Selection of packets can be computation-based. There appears to be no way to select packets by content using differentiating cryptography[9] alone.
– Authorization decisions can be computation-based, and secure since they are running inside secure hardware. In this case, even in the absence of a PKI to perform full authorization, an authorization decision can be made based on the number of packets in the result—no query with more than one matching packet will be authorized. Since computation must be performed to calculate such properties of the matching data set, this cannot be done securely without using secure hardware.

Performance. High performance was not one of the aims of our prototype, but we include some figures for completeness. The Encoder could process a 1.6 MB

[9] meaning that different sections of the stored archive are encrypted with different keys

packet dump to produce an archive in 6 seconds. A 630K dump took 2.3 seconds. A 2.0 MB dump failed due to insufficient memory. A Decoder run (without restrictions on packet number returned) on the 630K archive (1000 packets) which selected 105 packets took 6.3 sec.

Future optimizations will clearly have to focus on the Encoder, which in practice will have to keep up with a fast network.

5.2 Relation to the Advanced Packet Vault

One of the primary concerns of the Advanced Packet Vault project from Michigan [1] is speed—to enable the vault to keep up with a 100Mb network at high load. The major areas of concern are system questions of keeping packets flowing into their final long-term storage as fast as they are picked up. Since we do not, nor do we intend to, consider questions of system infrastructure, our work can combine well with the Advanced Vault project to produce a fast and more secure device. The 4758 Model 2/23 secure coprocessor can perform TDES on bulk data at about 20 MB per second, which is sufficient to keep up with the Advanced Vault system infrastructure on a 100 Mb network.

6 Conclusions and Future Work

6.1 Feasibility

We had several goals for our initial experiments. The community had long speculated on the use of secure coprocessors for computational enforcement of rights management. We carried this speculation one step further, by completing an implementation on a commercial platform, that could (in theory) be deployed on a wider scale with no additional technological infrastructure.

One of the purported advantages of the coprocessor approach to secure data processing is the ability to insert a computational barrier between the end user and the raw data. For this advantage to be realizable, however, this barrier must be able to support useful filter computation. By porting the `Snort` package inside the coprocessor environment, we experimentally verified this potential, for the application domain of archived network data.

6.2 Future Steps for Network Vault

We plan further work both within the application domain of archived network data, as well as in other domains.

With this domain, we immediately plan to expand our prototype along these major directions:

Performance. Currently, we limit the size of requests and archives to what can fit inside the coprocessor at one time. This limitation is unacceptable. We will augment the current arrangement to enable arbitrary-sized packet dumps and archives to be passed into the vault coprocessors. The upcoming release of Linux for the 4758 (as a replacement for CPQ/++) will make this easier, as Linux will provide more abstract host-card communication services, which will be useful in implementing a continuous data flow through the vault. When this is done, archive size will be decided by the long-term storage medium, like CD-ROM.[10]

We plan to attach the Encoder card to a live packet source (some sniffer) to enable live data collection.

Policy. Currently, the capabilities of the policy mechanism are very limited. The query language (Snort rules) is very flexible, but the macro limits and authorization procedures are very bland. We plan to implement limits on more parameters like packet quantities/rates and number of hosts involved. On the authorization side, the first thing would be to implement a history service—keep a log of accesses to enable for example a requirement like "packets can only be decrypted once". A full authentication of requests would depend on some authorization infrastructure, like those mentioned in Sect. 2.4. For post-processing, we will implement scrubbing functions to erase any packet parameters required.

Longer-term issues include setting up a policy administration system (for non-specialist managers), ensuring that access policy is consistent with overall goals, and exploring ways to allow access policy to existing archives to change, without violating existing policy promises.

Systems for authorization sooner or later assume that specific individuals or roles wield exclusive control over specific private keys. A serious part of a fully viable data vault would be a deployable infrastructure that makes this assumption true with reasonably high assurance. Ongoing campus PKI work at Dartmouth could be very useful here. [13]

As an early sketch of this project considered, [20] the binding of an archive to a single Decoder leaves the door open to denial of service attacks through Decoder destruction (just a tamper attempt will do the job). This binding is the current way of ensuring that the archive is accessed only through its attached policy, and there are no easy alternatives for achieving this assurance. Investigating the question will be important in making the armored vault practically acceptable.

6.3 Future Application Domains

The general framework of storing network packets securely and with an attached access policy, and using secure coprocessors to enforce this access policy applies

[10] A fully general solution to this problem broaches the issue of *private information retrieval*—since the user (who wants access to the data) may learn unauthorized things by observing how the coprocessor accesses the large archive. In related work, we have examined the use of coprocessors for practical solutions to this problem. [22]

to access rules for any large data set. We now discuss a few that we have been exploring.

Remote Data Storage. We have close ties with the Condor Project at the University of Wisconsin in Madison[11], who have expressed an interest in remote storage of large data sets with associated access controls. This problem falls squarely into our system of securing data with flexible and assured access policy.

If a researcher at site A wants to send data to site B, but ensure that the data is accessed only in accordance with some policy, she could proceed as follows. She uses a version of our Encoder to secure the data and attach her policy. She sends the data to site B, who have the Decoder to work with the data. They can gain only the kind of access site A specified. For example access could be limited to some specific research group, or the data can only be accessed a limited number of times, or some details about the data must always be hidden even if they are used in calculations inside the vault.

One coprocessor may suffice in this scenario, and it would have a different user interface than the armored packet vault (operating via an SSL connection to a client program perhaps), but the basic idea of computationally-expressed access control executed inside secure hardware remains.

For an immediate realization of this domain, consider the issues involved if site A wishes to release archived network data to site B, for use in "legitimate" research. How can the site A user community—and all other stake-holders—trust that these usage rules will be followed?

Academic PKI. Here in the Dartmouth PKI Lab, we have been interested in deploying an academic PKI that has at least some assurances that private keys remain private.

This domain raises many areas where we need to balance community interest against privacy interests. For example:

- The community may decide that certain administrative or law enforcement scenarios may require access to data that students have encrypted with their keys.
- Students may lose the authenticators (e.g. passphrases, or smart cards) that guard access to their private keys; one preliminary study [8] showed this was very common in segments of our undergraduate population.
- Students may encrypt data in the role of an organization officer—but then may leave before duly delegating access rights to their successor.

All of these scenarios require selective weakening of the protections of cryptography, should the community as a whole decide it is justified. Unlike standard approaches to key escrow and recovery, our armored vault approach permits tuning of access to exactly the community standard—and also provides defense against insider attack, since (if the policy and authorization are sound, and the Level 4 validation is meaningful), neither bribery nor subpoena can enable the vault operator to go beyond the pre-defined access rules.

[11] `http://www.cs.wisc.edu/condor/`

Appendix

Acknowledgments The authors are grateful to Charles Antonelli and Peter Honeyman at Michigan, John Erickson of HP Labs, and our colleagues in the Dartmouth and Wisconsin PKI Labs, for discussion and suggestions.

This work was supported in part by by the U.S. Department of Justice, contract 2000-DT-CX-K001, by Internet2/AT&T, and by IBM Research. However, the views and conclusions do not necessarily represent those of the sponsors.

References

1. Charles Antonelli, Kevin Coffman, J. Bruce Fields, and Peter Honeyman. Cryptographic wiretapping at 100 megabits. In *SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, Orlando, Apr 2002.
2. C.J. Antonelli, M. Undy, and P. Honeyman. The packet vault: Secure storage of network data. In *Proc. USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, April 1999.
3. Steven Bellovin, Matt Blaze, David Farber, Peter Neumann, and Eugene Spafford. Comments on the Carnivore system technical review. `http://www.crypto.com/papers/carnivore_report_comments.html`, December 2000.
4. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust-management system version 2. RFC 2704, `http://www.crypto.com/papers/rfc2704.txt`, Sept 1999.
5. Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proc. IEEE Conference on Security and Privacy*, Oakland, CA, May 1996.
6. Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the PolicyMaker trust management system. In *Financial Cryptography.* Springer, 1998.
7. Mike Bond and Ross Anderson. API-level attacks on embedded systems. *Computer*, 34(10):67–75, Oct 2001.
8. E. Etu and J. McIsaac. Bringing PKI to Dartmouth. Class Project, CS88, Dartmouth College, June 2001.
9. FBI. Carnivore diagnostic tool. `http://www.fbi.gov/hq/lab/carnivore/carnivore.htm`, Mar 2001.
10. Internet Engineeringt Task Force. Simple public key infrastructure (SPKI). `http://www.ietf.org/html.charters/spki-charter.html`, 1997.
11. Amir Herzberg, Yosi Mass, Joris Michaeli, Yiftach Ravid, and Dalit Naor. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P 2000)*, Berkeley, CA, May 2000. IEEE.
12. IBM. *IBM 4758 PCI Cryptographic Coprocessor Custom Software Interface Reference.* `http://www-3.ibm.com/security/cryptocards/html/library.shtml`.
13. N.F.S. Knight. College 'net security gets $1.5m. *The Dartmouth*, Feb 1 2002. `http://www.thedartmouth.com/`.

14. National Institute of Standards and Technology. Security requirements for cryptographic modules. Federal Information Processing Standards Publication 140-1, 1994.
15. Sandra Payette and Carl Lagoze. Policy-carrying, policy-enforcing digital objects. In J. Borbinha and T. Baker, editors, *ECDL 2000*, pages 144–157, Lisbon, Portugal, 2000.
16. Martin Roesch. Snort - lightweight intrusion detection for networks. In *13th Systems Administration Conference - LISA '99*. USENIX, November 1999.
17. Sean W. Smith and Steve Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31:831–860, 1999.
18. Stephen P. Smith, Jr. Henry H. Perritt, Harold Krent, and Stephen Mencik. Independent technical review of the Carnivore system. `http://www.usdoj.gov/jmd/publications/carniv_final.pdf`, Dec 2000.
19. S.W. Smith. Outbound authentication for programmable secure coprocessors. Technical Report TR2001-401, Department of Computer Science, Dartmouth College, March 2001. `http://www.cs.dartmouth.edu/~pkilab/oatr.pdf`.
20. S.W. Smith, C.J. Antonelli, and Peter Honeyman. Proposal: the armored packet vault. Draft, Sep 2000.
21. S.W. Smith, R. Perez, S.H. Weingart, and V. Austel. Validating a high-performance, programmable secure coprocessor. In *22nd National Information Systems Security Conference*. National Institute of Standards and Technology, October 1999.
22. S.W. Smith and D. Safford. Practical server privacy using secure coprocessors. *IBM Systems Journal*, 40(3), 2001. (Special Issue on End-to-End Security).
23. Statewatch. European parliament and EU governments on a collision course over the retention of data. `http://www.statewatch.org/news/2001/nov/15eudata.htm`, Nov 2001.
24. Dan Wallach. Copy protection technology is doomed. *Computer*, 34(10):48–49, Oct 2001.
25. Bennet Yee and J. D. Tygar. Secure coprocessors in electronic commerce applications. In *Proc. First USENIX workshop on Electronic Commerce*, New York, NY, July 1995.