# Cryptographic wiretapping at 100 Megabits

Charles J. Antonelli, Kevin Coffman, J. Bruce Fields, and Peter Honeyman

Center for Information Technology Integration, University of Michigan, Ann Arbor*

## ABSTRACT

This paper describes the Advanced Packet Vault, a cryptographically secured archiver of network packet data that reliably captures all packets on a 100 Mbps Ethernet network, encrypts them, and writes them to long-term magnetic tape storage for later analysis and evidentiary purposes. Based on a previous prototype, the APV provides an enhanced cryptographic organization that allows site-specific selection of the encryption format and that permits selected traffic to be made available without compromising the security of other traffic. The APV operates reliably under a continuous 100 Mbps load. We conclude with a discussion of future work necessary to scale the APV beyond 100 Mbps.

**Keywords:** security, privacy, evidentiary record, intrusion detection, packet sniffing

## 1. INTRODUCTION

The objective of the packet vault project at the Center for Information Technology Integration remains the production of a cryptographically secured long-term store of network packets for later use as input data for intrusion detection algorithms or for possible evidentiary purposes.

We describe the Advanced Packet Vault (APV), which reliably archives all of the network traffic on a 100 Mbps Ethernet subnet. Previous work at CITI includes the development of a prototype[1] and a 10 Mbps APV.[2]

As noted previously,[1] creating a complete and permanent record of all activity on a subnet addresses security threats by providing a corpus of data suitable for training and comparing intrusion detectors, detecting and helping to shape responses to intrusions in progress, providing a record of activity on a subnet, or if properly constructed, serving as evidence in legal proceedings.

The remainder of the paper is organized as follows. While the main foci of the APV project have been improved performance and reliability, the cryptographic organization of the APV has been significantly extended. Therefore, a brief review of the project goals is followed by a description of the new cryptographic organization. We then present the APV's architecture and discuss the hardware and software used, and present results of performance tests which establish the APV's ability to handle reliably any traffic on a 100 Mbps network. We conclude with a discussion of future work necessary to scale the APV beyond 100 Mbps.

Further author information: (Send correspondence to cja@citi.umich.edu)
Center for Information Technology Integration, University of Michigan, 535 W William St, Ann Arbor, MI 48103-4943

| Type | Advertised MBps | Measured MBps | Native GB/volume | $/GB | cc/GB |
|---|---|---|---|---|---|
| AIT2 | 6 | | 50 | 1.50 | 1.9 |
| DLT8000 | 6 | 4.6 | 40 | 1.40 | 7.0 |
| Mammoth2 | 12 | 9.6 | 60 | 1.50 | 1.5 |
| AIT3 | 12 | | 100 | 1.29 | 0.95 |
| LTO | 15 | 13 | 100 | 1.15 | 2.3 |
| Super DLT | | | 110 | 1.05 | 2.6 |
| DVD-R | 2.76 | | 4.7/9.4 | 0.47-0.63 | 3.2/1.6 |
| EIDE | 17-35 | | 163 | 1.57 | 2.38 |

**Table 1**: Comparison of Storage Technologies.

## 2. GOALS

We retain our goals of commodity, completeness, and security from our prototype[1]:

**commodity:** We have built the APV from high-performance commodity hardware and software, continuing to avoid expensive or special-purpose platforms and to depend on Moore's Law to deliver increased capacities down the road.

**completeness:** As any attempt at packet triage can be exploited by an adversary, we defend against such attacks by building a vault that can reliably archive every packet seen on the network. The APV is designed to archive all traffic on a 100 Mbps network under any conditions, including attacks against the vault itself.

**security and privacy:** We use cryptography to protect privacy so that a breach of physical security does not expose the data contained in an archive. We use strong cryptography with strong keys, we change keys periodically, and we use keys that are dependent on the IP addresses on each packet, so that some subsets of the data can be revealed without disclosing others.

**openness:** It is difficult to convince users that a system respects their privacy if its operation is opaque. Source code for all of our software (including the Linux operating system) is freely available. The vault is designed to operate reliably and securely despite the existence of adversaries with complete knowledge of its design.

The prototype vault used CD-ROM for its permanent record. While we could have used this medium for the APV, future editions of the vault will be expected to operate at 100 Mbps and above. When attached to a heavily loaded 100 Mbps network, the challenge is to capture, process, and store about a terabyte each day.

We are sensitive to the recurring costs of operation, which include personnel costs for system operation and maintenance, storage costs for media, and the cost of media. Our targets are to store a year's worth of 100 Mbps vault data in a cubic meter, at a cost of $50,000 for physical media. These targets translate into $0.135 and 2.7 cc per gigabyte.

Today's LTO tape technology meets our volumetric target and costs $1.15/GB. While this is down from $1.50/GB one year ago, it is a bit off the pace predicted by Gray and Shenoy,[3] who suggest that storage cost is improving by a factor of four every three years. If the prediction proves accurate, this yields an annual cost for media of almost $140,000, a dominating and forbidding price tag. Under certain assumptions, though, such as compressibility of the raw network traffic, volume discounts for tape cartridges purchased by the thousands, and the emergence of unconventional storage media such as optical tape, we anticipate this cost to fall by an additional factor of two to four, which achieves our cost target. See Table 1 for a comparison of storage media.

Advances in magnetic disk technologies threaten to obsolete tape altogether. Although based on the same fundamental technology, market forces and economies of scale have altered the conventional 10:1 storage cost ratio advantage of tape over disk so that it is today only about 3:1.[3] Indeed, Table 1 shows that the advantage is nearly negligible today. Furthermore, technologists are predicting two orders of magnitude improvement in disk densities over the next five years,[4] which approaches our cost objective and far surpasses our volume goal.

Optical technologies are also becoming viable. We abandoned CD technology because it would require over a thousand CDs to store our anticipated daily terabyte. Yet, the table shows that DVD is in the ball park. Tomorrow's Blu-ray DVDs will store 27 GB,[5] and the DVD Forum is discussing technologies that might double or quadruple that. If writable Blu-ray blanks can be obtained for a few dollars, then our cost and volume goals can be met with this technology.

These developments in storage systems have caused us to modify a goal:

**permanence:** Despite consistently bad long-term experiences with data storage on magnetic tape and recent improvements in density and cost in the DVD arena, magnetic tape remains the medium that best meets our speed and density targets. But this is changing fast; we expect optical drives to compete and magnetic disk drives to dominate within a few years.

## 3. CRYPTOGRAPHIC ORGANIZATION

The APV's cryptographic organization is heavily influenced by the security and privacy goal. In the event of loss of physical control over the vault, we must rely entirely on the strength of the cryptography for the security of the vault archives. Also, in order to allow use of the vault data with the minimum disclosure necessary, the vault is designed so that cryptographic keys can be given out which decrypt only subsets of the archived data.

As in the prototype, we use a symmetric cipher to encrypt packet data, and an asymmetric *master key* to encrypt the symmetric keys. The master public key is stored in the APV and is used during packet collection; the encrypted symmetric keys are written to tape along with the encrypted packets. When retrieving packets from the APV, the master private key is used to recover the symmetric keys, which are used to decrypt the retrieved packets. We refer to the agent possessing the private key as the vault *owner*.

The priorities that shape the vault's cryptographic design are as follows.

**Privacy:** It should be infeasible for an adversary without possession of any keys to recover any data, or for an adversary with some keys to discover other keys. Given a request for data from the vault, it should be possible to generate keys that decrypt the requested data without exposing other data.

**On-line performance:** The vault must be able to keep up with a 100 Mbps network at all times.

**Off-line performance:** The vault should be able to answer common types of requests for stored data in a reasonable amount of time.

While we won't give up privacy or on-line performance to gain off-line performance, there are ways that off-line performance can affect privacy. For example, if the vault is not designed to handle common kinds of queries easily, then recovery requests may be overly broad. Also, we prefer to minimize the complexity of the work that must be done with the vault master key, to minimize accidental disclosure or misuse.

### 3.1. General organization

The vault gathers incoming packets into *segments*, which are in turn gathered into *volumes*. Segments are at most 16 MB in size and are encrypted and aggregated into 1 GB volumes, which are written to tape. The vault only stores those Ethernet frames that contain IP packets; all others are discarded.

With each new volume, two symmetric keys are created, a *volume key* $K_V$ and a *translation table key* $K_T$. These keys are created using a strong random number generator and are used to encrypt the packet data in each segment, as described below. After a volume is completed, both keys are encrypted under the vault master public key and written to tape.

vault open header format:

| | | | | |
|---|---|---|---|---|
| length | timestamp | $E_{K_C}$(Ethernet header) | translated IP header | $E_{K_C}$(IP payload) |

conversation format:

| | | |
|---|---|---|
| length | $E_{K_T}(K_C)$ | $E_{K_C}$(timestamp, Ethernet header, IP packet) |

endpoint format:

| | | | | |
|---|---|---|---|---|
| length | $E_{K_S}(K_C)$ | $E_{K_D}(K_C)$ | $E_{K_T}(K_C)$ | $E_{K_C}$(timestamp, Ethernet header, IP packet) |

**Figure 1.** The formats of encrypted packets offered as alternatives by the vault; $E_{K_S}$, $E_{K_D}$, $E_{K_C}$, and $E_{K_T}$, represent encryption under source and destination endpoint keys, conversation keys, and translation table keys, respectively.

The volume key is used exclusively to generate *conversation keys*, which in turn are used to encrypt packets. The conversation key used to encrypt a packet depends both on the volume key and on the source and destination address in that packet.

Note that finding the correct volume key(s) for a requested time interval will require reading through the vault output unless the vault owner maintains a database mapping epochs to volumes and encrypted volume keys. The APV keeps such a database on local disk, although the current version does not include the volume keys.

## 3.2. Encryption formats

The APV supports three encryption formats as described below. All formats generate a conversation key $K_C$ derived from the packet's source and destination IP addresses as follows

$$K_C = E_{K_V}(\texttt{src}||\texttt{dst})$$

In other words, we compute $K_C$ by concatenating the IP source and destination addresses and encrypting the result using $K_V$.

### 3.2.1. Open Header format

The *open header* encryption format uses $K_C$ to encrypt the link-level header and body of each IP packet under the conversation key, but does not encrypt the IP header; instead, the IP addresses in the header are translated using a *translation table* built on-the-fly for each segment. We then encrypt the translation table using $K_T$. This was the only format supported by the prototype vault.[1]

A corpus of cleartext packet headers is useful for research and for a variety of applications, including traffic analysis.[6–8] Much of the fundamental Internet protocol performance work has depended critically on the availability of packet header traces. Open header format allows the vault archives to deliver packet headers while protecting packet contents.

Several issues arise, however. A simple mapping of IP addresses may not be sufficient to obscure information about which hosts were involved in a given conversation. For example, an attacker could expose the mapping used for a particular IP address by sending a recognizable packet across the network, with a spoofed source address equal to the address he or she wishes to discover. Mappings also could be exposed by observing unusual IP header fields or packet lengths.

It is also possible to identify all packets of a conversation within the same segment by observing that they all possess the same translated address. This identification does not cross segment boundaries, as a new translation table is built for each segment, and each translation table is encrypted in CBC mode with a different initialization vector.*

Finally, managing the translation table efficiently is difficult in the face of sustained and determined attacks. For example, injecting packets with unique spoofed source addresses forces the translation table to grow and defeats conversation key caching.

*The prototype vault encrypted the tables in ECB mode, so that it was possible to identify connections between conversations across segment boundaries.

#### 3.2.2. Conversation format

To address the above concerns we have added a new encryption format to the cryptographic organization. *Conversation format* encrypts each packet in its entirety under $K_C$, including the IP header. This also removes the need to maintain a translation table. However, it is now infeasible to recover an entire volume of data, because it is impossible to know which conversation key to use to decrypt a given packet without knowing its source and destination addresses *a priori*. To make volume recovery practical, we prepend a copy of $K_C$ encrypted under $K_T$ to each packet.

#### 3.2.3. Endpoint format

A third encryption format generates two additional *endpoint keys* $K_S$ and $K_T$ for each packet: $K_S$ depends only on the source address and $K_V$; $K_T$ depends only on the destination address and $K_V$. Two additional copies of $K_C$ are prepended to each packet, one encrypted with each endpoint key. This allows someone in possession of an endpoint key to decrypt all traffic which was sent or received by a particular host.

Creating files in endpoint format consumes more processing time and increases the per-packet storage overhead, but it allows the vault to satisfy queries for all traffic sent to or received by a given host by decrypting only the designated traffic, rather than resorting to decrypting entire volumes.

The layout of a packet stored in each of these formats is summarized in Figure 1.

### 3.3. Key generation

Conversation and endpoint keys are generated by encrypting the IP address or addresses using the volume key.

An analysis of the algorithm used by the prototype vault revealed a flaw that made it easier than it should have been for an adversary in possession of one key to find other keys. That algorithm was also designed specifically for use with DESX,[9] and didn't obviously generalize to algorithms with different block or key lengths.

Therefore a new algorithm was required, meeting the following requirements:

**Repeatability:** To allow easy retrieval, the algorithm must use only knowledge of the IP addresses involved and the volume key; the key generated shouldn't depend on other random numbers or past history of the vault.

**Security:** It must be hard to guess the volume key given only knowledge of some addresses and corresponding conversation keys, and it must be hard to guess conversation keys or parts of such keys given only knowledge of addresses or other conversation keys.

**Flexibility:** The key-generating algorithm should take a variable amount of input: we might decide to index on something other than source and destination addresses; we might also use port addresses, or might use some other kind of address (e.g. Ethernet MAC addresses or IPv6 addresses). The algorithm should also be able to produce variable amounts of output, so that we can easily change the lengths of the keys produced.

**Speed:** The algorithm must be fast enough to meet the performance requirements under all conditions. If this is not the case, and if we depend on key caching for adequate speed, then an attacker could disable the vault by sending numerous packets with spoofed IP addresses. The vault is robust against such an attack only if the key-generation algorithm is fast enough to generate a new key for every packet.

To generate conversation keys, we concatenate the source and destination addresses to form 64 bits of plaintext that are encrypted with $K_V$. Endpoint keys are made by concatenating the source or destination address, 32 zero bits, and a unique byte; the resulting plaintext is encrypted with $K_V$. The unique byte ensures that duplicate conversation and endpoint keys are never generated. See Appendix A for a detailed description of the key generation algorithm.

Based on our experiments, we believe that our new algorithm meets all requirements. However, the final requirement seems to be the most difficult; while we have met our performance requirements for the current generation vault, key generation and key scheduling may prove to be a bottleneck when we scale beyond 100 Mbps, and some form of key-schedule caching may be required.

### 3.4. Encryption algorithms

As we have seen, the vault requires (at least) two ciphers: a public-key cipher to use for the master key, and a symmetric-key cipher for everything else. Our criteria for choosing ciphers include:

**Security:** The completeness and permanence of the vault data demands encryption that will resist attack for as long as possible. This is particularly important for the private key, since it is not routinely changed, and since knowledge of that key alone is sufficient to expose all of the vault data.

**Performance:** This is less important for the public key cipher, since it is only used once per volume to encrypt a small amount of data. The symmetric cipher, however, must be able to handle the full 100 Mbps flow of data from the network. The performance of the key-scheduling algorithm is particularly critical, because in the worst case a new key is used for each packet.

**Availability:** We restrict our choice to freely-available implementations.

For symmetric encryption we use Rijndael, which has been selected by NIST as the new Advanced Encryption Standard (AES).[10] Rijndael provides variable block and key sizes; we use 128 bits for both, as these are convenient for the key-generation algorithm previously described.

A brute-force search of the 128-bit key space is likely to remain impractical for some time. Exactly how long depends mainly on guesses about progress in the computing hardware. Lenstra and Verheul estimate that a brute-force attack against a symmetric algorithm using 128-bit keys should remain infeasible for well over 50 years[11]; see also the discussion by Blaze et al,[12] who recommend 90-bit keys to protect information for the next 20 years.

This assumes no breakthroughs in the cryptanalysis of Rijndael. However, Rijndael did withstand a great deal of scrutiny during the the AES selection process, and its selection as the AES will guarantee that it is the target of more extensive cryptanalysis in the near future. If it is insecure, at least we are likely to find out sooner rather than later.

The performance of Rijndael is particularly well-suited to the vault; in addition to excellent performance on regular encryption, the key-scheduling algorithm is particularly fast.[13,14] For the 100 Mbps vault, we use Gladman's assembler implementation of Rijndael.[15] On the vault hardware, we have found this implementation can encrypt data at 250 Mbps even for small packets, with a key-scheduling operation taking less time than encryption of a single 128-bit block.[†]

The vault's master key is a 4096-bit ElGamal key, and we depend on GPG[17] to perform ElGamal encryption. Estimates based on publicly-available cryptanalysis[11,18] would lead us to believe this is a longer key than necessary. However, since the master key is only used to encrypt a small, fixed amount of data once per volume, the cost of this encryption is amortized over a volume's worth of data, which gives us considerable flexibility to choose cryptography that may be stronger than required.

Finally, as noted in our previous work,[1] exposed vault data continue to pose a problem for any encryption scheme. There is no way to guarantee that encrypted data will withstand advances in cryptanalysis or technological improvements over arbitrarily long periods of time. If the period is sufficiently long, it may be that a given vault's data is no longer valuable enough to be worth the effort of decryption, or sensitive enough for its exposure to be a concern. These risks will have to be evaluated as more vaults are deployed.

[†]The raw encryption speed of Gladman's implementation on our hardware is 350 Mbps, a marked improvement over the 200 Mbps observed for an optimized ANSI C implementation[16] used in the 10 Mbps vault.

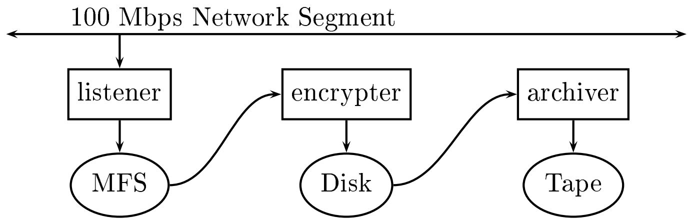


**Figure 2**: Vault data flow

## 4. ARCHITECTURE

Since the development of the vault prototype, advances in commodity hardware made it possible to consider collapsing the vault onto a single machine. Accordingly, the main design consideration for the APV was to perform all tasks on a single host.

In addition, advances in magnetic tape technology, embodied in the commercial availability of LTO Ultrium tape drives and media, now permit the storage of 100 GB of data at 15 MBps (nominal) on a tape cartridge. Despite misgivings about longevity and reliability, we have chosen magnetic tape as the most cost-effective solution for the APV archives.

The APV hardware consists of a machine built from an Intel STL2 Server Board containing two 866 MHz Pentium III processors with 133 MHz system bus speed and 512 MB of system memory. This motherboard contains an integrated 10/100 Mbps Fast Ethernet Controller. Two high-speed 35 GB Ultra160 SCSI disk drives are attached to an on-board Ultra160 SCSI controller to buffer data after it has been collected and encrypted. A single EIDE disk contains the operating system partitions. APV data are stored on a Qualstar 8211 Tape Library Subsystem. The library contains an HP Model 230 LTO Ultrium tape drive and a cartridge picker; both devices are attached to the same Adaptec 29160 SCSI controller.

We evaluated several open-source operating systems including OpenBSD, FreeBSD, and Linux. Our major criterion was adequate performance across all stages of the packet data pipeline. We chose Linux because of its memory-mapped kernel packet interface,[19] its built-in support for multiprocessing and software RAID, and its high-performance SCSI tape driver, which sustains writes at 13 MBps.

We used OpenBSD for our 10 Mbps vault. While its reputation for providing a secure operating system platform and excellent support for cryptographic hardware is of advantage, it does not as yet support multi-processing. In effect, we traded special-purpose hardware for software encryption; this is a good tradeoff as the second processor is useful for tasks other than encryption.

The APV software consists of three major components, described below, driven by two scripts. A script called *pilot.pl* drives two of the components, the *listener* and *encrypter*. A second script called *archiver.pl* drives the third component, the *archiver*. See Figure 2 for a data flow diagram.

Each component operates as follows:

**listener:** The kernel places packets received on a network interface in a ring buffer shared with a user space listener process, which periodically writes packets to a file in a memory file system (MFS). When the destination MFS file exceeds 16 Megabytes, or when more than a minute passes, the listener renames the file and starts a new file. We call each resulting raw packet file a segment.

**encrypter:** The encrypter reads each new segment from MFS, encrypts it according to the specified format, and writes the result to the SCSI raid partition. When 1 GB of segments has been collected, the encrypter starts a new volume.

**archiver:** The archiver encrypts the volume key and translation table key of each completed volume using the vault's public master key, and writes the result, together with all of the encrypted segments, to tape.

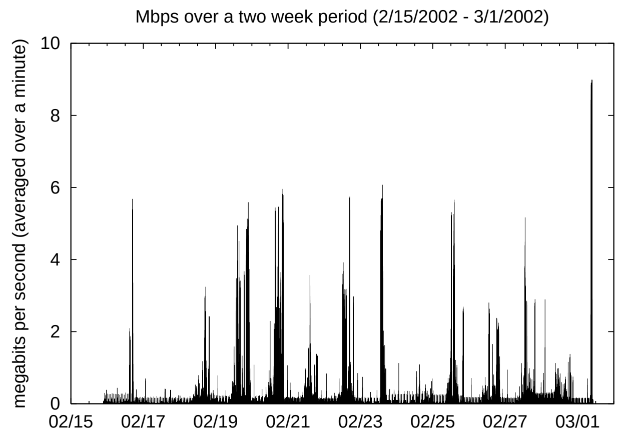


**Figure 3**: Observed Mbps during a 2-week period

We use the Linux built-in strong random number generator to generate volume and translation table keys; use of `/dev/random` effectively prevents the possibility of inferring future key values from previous ones. RAID in software is used to achieve adequate disk performance.

The pilot script starts the listener and invokes the encrypter program for each completed segment. The archiver script waits for completed volumes and writes them to tape using the `tar` archive utility. The archiver also manages the tape library, prompting for more tapes when all the tapes in the library have been filled.

We use a separate program `apvsync` to synchronize the activities of the other components using shared memory and semaphores. We extend the classic producer-consumer synchronization model by allowing a consumer to return an item for future consumption if it cannot be processed due to some transient error. When a volume is completed, the pilot script uses `apvsync` to synchronize with the archiver.

This architecture permits the APV to use multiple tapes drives simultaneously. We have verified operation with two tape drives. While a single LTO drive is sufficient for 100 Mbps operation, multiple drives will be necessary for scaling above this rate.

The archiver maintains a database of all the volumes archived. This database can be examined at a later time to find volumes of interest for restoration. The database retrieval interface is currently a manual procedure, but an automated mechanism is planned.

## 5. PERFORMANCE TESTING

Since the vault encrypts each packet under a key depending on both the source and destination addresses, there is a great deal of overhead associated with each new packet. To write a packet to a file in the conversation format, the vault has to examine the addresses in the packet and generate a conversation key, encrypt a copy of the new conversation key under the translation key, and then create a schedule for the conversation key, all before it can begin encrypting the packet. To write a packet in endpoint format, the vault must, in addition, generate and schedule both endpoint keys and use both of them to encrypt the conversation key. When using open header format, the vault avoids having to prepend encrypted keys to each packet, but must instead perform translation of IP addresses.

Only a restricted subset of the possible IP addresses will appear on a network under ordinary conditions, so it is likely that great performance improvements could be made by caching conversation and endpoint keys. However, an adversary might be able to inject packets with spoofed addresses onto the network in order to defeat this caching. Also, the adversary could make those packets as small as possible, forcing the vault to spend most of its time doing key generation and scheduling.

| Packet Size | Max Input Rate (observed) | Max Input Rate (calculated) | Max Output Rate (calculated) |
|---|---|---|---|
| 60 bytes | 44 Mbps (92K pps) | 68 Mbps (142K pps) | 65 Mbps (135K pps) |
| observed distribution | 70 Mbps (39K pps) | 83 Mbps (46K pps) | 93 Mbps (52K pps) |
| 1514 bytes | 93 Mbps (8K pps) | 94 Mbps (8K pps) | 104 Mbps (9K pps) |

**Table 2**: Performance extrema. Observed rates are lower bounds, as our test harness could not fully load the APV.

For maximum robustness, therefore, we have designed the 100 Mbps vault to keep up with a fully loaded network without any caching of keys or key schedules.

### 5.1. Actual traffic measurements

We ran the vault continuously for two weeks while monitoring all of the traffic on the CITI local area network. The traffic levels during this period are shown in Figure 3. A total of 29 GB and 66 million packets were stored to tape by the vault. The vault was using conversation format. No packets were dropped during this period.

### 5.2. Synthetic traffic measurements

As the load imposed on the vault by actual traffic was quite low, tests with synthetic data were conducted by attaching the vault to a private 100 Mbps Ethernet hub shared with several other machines that could produce packets of chosen rates and sizes. The vault was using conversation format. Using this test harness and a program designed to output packets at controlled rates and sizes, we were not able to force the vault to drop packets or to otherwise fail. We were able to generate a maximum load of 93 Mbps using 1514 byte packets.

At high loads, the vault can fail in two ways: if the encrypter falls behind, then MFS will overflow; and if the tape drive falls behind, then the on-disk buffer in UFS will overflow. Table 2 shows maximum rates at which the vault can perform without overflowing these buffers, under three different test loads:

60 byte packets: Traffic consisting completely of the smallest frames allowable on an Ethernet (14 bytes of header, and 46 bytes of payload).

observed distribution: Traffic with a combination of four packet sizes, chosen to correspond roughly with the peaks in distribution of packet sizes seen on our local network during an earlier run of the vault,[2] but erring on the side of smaller packets, which stress the vault more for a given Mbps load. More specifically, 36% of the packets were 60 bytes long, 53% were 159 bytes long, 5% were 598 bytes long, and 6% were 1478 bytes long.[‡]

1514 byte packets: Traffic consisting completely of the largest frames allowable on an Ethernet.

The "observed" column of Table 2 shows the maximum observed packet rates we obtained using our test harness; for the 60-byte and observed distribution loads, our harness was not able to fully load the APV. Note that the rates reported here do not include additional components of network overhead, including checksums, preambles, and interframe gaps, which are required to transmit each packet at the physical layer, but which are not seen by the vault software. The "input" column gives the maximum rate at which packets can be delivered by a 100 Mbps network, taking into account this overhead. The "output" column shows the maximum input rate at which the vault could store packets without developing a backlog, assuming that the tape drive is the bottleneck of the data pipeline. We are not entirely convinced of this assumption, but a higher-speed test harness will be required to probe further.

[‡]Note that this distribution has a much lower proportion of large packets than that observed in a more recent run of the vault (see Figure 4), which covered a period of higher traffic on the CITI network.

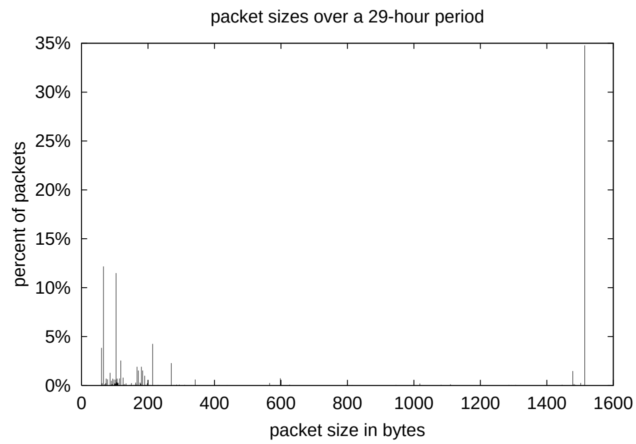


**Figure 4**: Observed distribution of packet sizes as a percentage of total packets

It can be seen that except for the case consisting entirely of packets of the smallest possible size, the vault capacity exceeds the requirements. In any event, we are uncertain whether the calculated input rates are achievable in practice.

### 5.3. Open Header format traffic measurements

We tested the APV on traffic consisting of packets of the minimum allowable size (60 bytes) at the maximum possible rate, with each packet having distinct spoofed IP addresses. This forces the generation of new conversation keys at a high rate, and places maximum load on an APV processing packets using the open header format. On a real network, this sort of traffic would occur only as the result of a concerted attack on the APV by an adversary with direct access to the network. Again, we were unable to force the vault to lose any packets.

## 6. DISCUSSION AND FUTURE WORK

We have constructed a packet vault using commodity parts that fully meets its operational goal of encrypting and archiving all packets seen on a 100 Mbps network.

With this APV in hand, we plan to investigate the landscape beyond 100 Mbps. There are many opportunities for future work.

One way to improve the APV's reliability and scalability is to design the APV to handle a wide range of "normal" traffic well, and to back off gracefully in the event of a denial of service attack. The APV already has mechanisms in place to monitor its resources, such as free space on filesystems where packets are buffered, but ceases operation immediately when resources become exhausted. We plan to investigate alternative failure modes, such as switching to bulk encryption under a single key to avoid the overhead of generation and scheduling of conversation keys.

We plan to continue to investigate alternative cryptographic organizations that give APV owners the flexibility to implement policies that balance performance, privacy, and the production of useful data. Further investigation is also needed to determine how best to disclose information such as headers for research purposes with minimal risk of disclosing personally identifiable data. In particular, further work is necessary on scalable methods for mapping IP addresses, and we may also consider encrypting headers under keys distinct from those used to encrypt packet payloads.

We will continue to monitor developments in cryptographic hardware. For example, the Hifn 8154 chip reportedly encrypts even small packets at gigabit speeds.[20] However, we must pay close attention to the performance penalties incurred by large numbers of encryption keys. We will also watch DVD-R hardware; improvements in recording rates and availability of commodity jukeboxes may allow the APV to return to write-once media with a better longevity track record than that of magnetic tape.

Finally, gigabit Ethernet is becoming widely deployed, and multi-gigabit backbones are no longer exceptional. A single APV cannot cope with these network speeds. We will investigate a parallel architecture in which groups of APV engines cooperate to cover a high-speed network, possibly using a round-robin technique or a CRC of a packet's contents to distribute packets among multiple engines.

## ACKNOWLEDGMENTS

Our work on the 10 Mbps APV was supported by the Department of Justice under Dartmouth College Sponsor Award No. 2000-DT-CX-K001.

## REFERENCES

1. C. J. Antonelli, M. Undy, and P. Honeyman, "The packet vault: Secure storage of network data," Tech. Rep. 98-5, Center for Information Technology Integration, 535 W. William St., Ann Arbor, MI, 1998.
2. C. J. Antonelli, K. Coffman, and J. B. Fields, "The 10 Mbps advanced packet vault," Tech. Rep. 01-10, Center for Information Technology Integration, 535 W. William St., Ann Arbor, MI, 2001.
3. J. Gray and P. Shenoy, "Rules of thumb in data engineering," Tech. Rep. MS-TR-99-100, Microsoft Research, Redmond WA, 2000.
4. M. H. Kryder, "Future magnetic recording technologies." `http://www.usenix.org/events/fast/kryder.pdf`, 2002. Invited talk at FAST 2002, Monterey.
5. Sony Corp., "Large capacity optical disc video recording format "Blu-ray disc" established." `http://www.sony.co.jp/en/SonyInfo/News/Press/200202/02-0219E/`, 2002.
6. A. Feldmann, A. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proceedings of ACM/SIGCOMM*, 1999.
7. J. van der Merwe, R. Caceres, Y. hua Chu, and C. Sreenan, "mmdump: A tool for monitoring internet multimedia traffic," Tech. Rep. 00.2, AT&T Labs Research, 2000.
8. "Macroscopic visualisation of the internet during october, 2000." `http://www.caida.org/analysis/topology/as_core_network/AS_Network.xml`, 2000.
9. J. Kilian and P. Rogaway, "How to protect DES against exhaustive key search," in *Advances in Cryptology – Crypto "96*, *Lecture Notes in Computer Science* **1109**, pp. 252–267, Springer-Verlag, 1996.
10. N. I. of Standards and Technology, "AES home page." `http://csrc.nist.gov/encryption/aes/`.
11. A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," in *Proceedings of PKC 2000*, *Lecture notes in Computer Science* **1751**, pp. 446–465, Springer-Verlag, 2000.
12. M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener, "Minimal key lengths for symmetric ciphers to provide adequate commercial security." `http://theory.lcs.mit.edu/~rivest/bsa-final-report.txt`, 1996.
13. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Fergesun, "Performance comparison of AES submissions," in *Proceedings of the Second AES Candidate Conference*, 1999.
14. B. Gladman, "AES algorithm efficiency." `http://fp.gladman.plus.com/cryptography_technology/aes/`.
15. B. Gladman, "Implementation of AES (Rijndael) in assembler." `http://fp.gladman.plus.com/cryptography_technology/rijndael/`.
16. V. Rijmen, A. Bosselaers, and P. Barreto, "Optimised ANSI C code for the Rijndael cipher." `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndael-fst-3.0.zip`.
17. "The GNU privacy guard." `http://www.gnupg.org/`, 2001.
18. R. D. Silverman, "A cost-based security analysis of symmetric and asymmetric key lengths," *RSA Laboratories Bulletins* (13), 2000.
19. "libpcap patches demonstrating use of memory-mapped packet sockets." `http://ftp.sunet.se/pub/os/Linux/ip-routing/lbl-tools/`.
20. Hifn Inc., "HIPP II - world's first gigabit security processor with onboard public key." `http://www.hifn.com/products/8154.html`, 2002.

# APPENDIX A. KEY GENERATION ALGORITHM

In this appendix, we describe the key generation algorithm in more detail:

Assume the following are given:

**X:** a block cipher (e.g., DESX, triple-DES, or AES)

**b:** the length of the blocks that X operates on

**D:** some data that the key depends on (e.g., a pair of IP addresses and port numbers)

**d:** the length in bits of $D$

**k:** the desired length in bits of the resulting key

Perform the following steps:

1. Let $m = d/b$ rounded up; this is the number of blocks required to represent $D$.
2. Let $n = k/b$ rounded up; this is the number of blocks required to represent $k$.
3. Let $p = m + n - 1$; this is the number of encryptions we'll perform.
4. Pad $D$ with $p * b - d$ zero bits, and call the result $D'$. (So $D'$ will have length $p * b$.)
5. Encrypt $D'$ using the volume key and $X$ in CBC mode with a zero initialization vector. Call the result J; it should be p*b bits long.
6. Discard the first $p * b - k$ bytes of $J$, and use the leftover bits as the payload key.

The resulting key depends only on the volume key and the conversation data, as required. Clearly, we can also produce variable amounts of input and output data as necessary.

To generate conversation keys, we take the data $D$ to be the concatenated source and destination addresses. Endpoint keys are made by concatenating the source or destination address, 32 zero bits, and a single byte representing either the character “S” or “D”. This extra byte ensures that duplicate conversation and endpoint keys are not generated.

The security of the algorithm rests on the security of the underlying block cipher $X$. Knowing many pairs $(D, K_D)$ is essentially the same as having many ciphertext/plaintext pairs under $X$. So recovering the volume key given many such pairs is as hard as a known-plaintext attack against the block cipher $X$. Also, guessing a conversation key $K_D$ based on the data $D$ is equivalent to being able to encrypt $D$ under the volume key without knowing the volume key. Since we use CBC and throw out the first $n - 1$ blocks, obtaining even a single bit of $K_C$ should require being able to encrypt every bit of $C$. Again, doing this should be as difficult as breaking $X$.