MICSS: A Realistic Multichannel Secrecy Protocol

Devin J. Pohly
SIIS Laboratory
Pennsylvania State University
University Park, PA 16802
Email: djpohly@cse.psu.edu

Patrick McDaniel
SIIS Laboratory
Pennsylvania State University
University Park, PA 16802
Email: mcdaniel@cse.psu.edu

***Abstract*—Flaws in cryptosystem implementations, such as the Heartbleed bug, render common confidentiality mechanisms ineffective. Defending in depth when this happens would require a different means of providing confidentiality, which could then be layered with existing cryptosystems. This paper presents MICSS, a network protocol which uses multichannel secret sharing rather than encryption to protect data confidentiality. The MICSS protocol ensures perfect secrecy against an $(n-1)$-channel attacker and operates at line speed in a three-channel throughput benchmark. MICSS provides a practical means of securing network communications, and it layers seamlessly with cryptosystems to mitigate the effects of implementation flaws.**

***Index Terms*—Secret sharing, multichannel, multipath, network protocols, confidentiality.**

I. Introduction

Cryptosystem implementations, as realized in current software, are not perfect. Even well-designed cryptosystems with mathematically sound algorithms have been completely undermined by failures in implementation or execution [1]. Recent flaws such as the OpenSSL Heartbleed vulnerability [2] serve as poignant reminders of this fact; in this case, an implementation defect exposed primary keying material, allowing an adversary to defeat any encryption that used those keys. Yet in spite of these failures, network communications are commonly protected solely by cryptosystem implementations. What is needed is a fundamentally different approach which can be combined with cryptography to defend in depth against implementation flaws and other cryptosystem weaknesses.

This work presents a new *multichannel secret sharing protocol*, MICSS, which augments an existing network stack with a second approach for defending confidentiality in depth. MICSS is based on the fact that there are frequently multiple channels over which two hosts can communicate. These channels may be networks with different physical characteristics, such as the Wi-Fi and mobile broadband connectivity found in mobile phones, or the wired and wireless interfaces on a laptop. They may even be multiple paths in one network, accomplished using proxy servers or VPNs to ensure routing via a given host. MICSS exploits the presence of these multiple channels to provide confidentiality. Given a message M and n channels, MICSS applies secret sharing [3], [4] to spread the message over shares $S_1, \ldots, S_n$, such that all n shares must be known to reconstruct M. One share can then be sent on each channel, forcing an adversary to compromise every channel in order to learn any part of the message. Figure 1 gives a detailed overview: even with an eavesdropper on two of the four channels in the figure, MICSS allows hosts A and B to communicate confidentially.

The abstract notion of multichannel secret sharing, however, is not usable on its own. Nor is it straightforward to adapt this idea for use in real networks. It requires new algorithms for sending and receiving messages, as well as a process for securely associating multiple channels which cannot be manipulated by an adversary. MICSS is the first real protocol which addresses these problems. It ensures perfect secrecy against an $(n-1)$-channel attacker, and it does so without significant loss of throughput. In addition, MICSS can be used "out of the box," requiring no modification to applications, networks, or the kernel. In microbenchmarks measuring raw throughput and latency, MICSS is capable of achieving throughput within 1% of line speed, with acceptable latency. It also performs well in web server macrobenchmarks, able to maintain heavy request loads as the number of concurrent clients increases, with overhead that is generally less than 5%.

II. Related Work

A. Secret Sharing and Information Theory

Secret sharing is a mathematical algorithm for spreading a piece of secret data across multiple "shares" such that a certain number of these shares are required, in any combination, to reconstruct the original data [3], [4]. It was invented independently by Shamir and Blakley in 1979. The following year, Blakley noted that one-time pads bear more similarity to secret sharing than to cryptography [5]. In the same paper, he introduced the notion of operating a secret sharing scheme in "courier mode," where it is used to safeguard a message carried by several couriers rather than a key shared among several individuals. The advantage of this proposal is apparent in light of Shannon's proof that one-time pads provide perfect information-theoretic secrecy [6]: an adversary who cannot capture a sufficient number of shares gains no information about the message. This idea is the crux of multichannel secret sharing, but Blakley himself went no further than noting the possibility of this application.

Later research in secret sharing built theoretically on this possibility, assuming the existence of a multichannel secret sharing protocol for support. For example, Yamamoto gave proofs bounding the key rate required for a given level of security (in terms of equivocation) for the two- and three-share

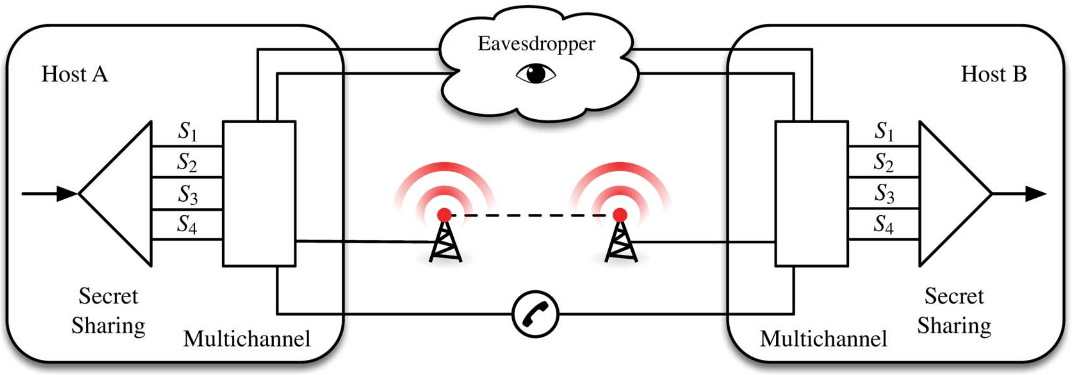


Fig. 1. Detailed overview of MICSS

versions of the scheme [7]. More recent work has designed routing algorithms to keep shares as far apart as possible [8], or proposed complex methods for splitting data among paths [9].

B. Secure Multichannel Protocols

The problem with these secret sharing works is that they assume the availability of usable multichannel protocols which provide confidentiality using secret sharing, when in reality *none have existed* until now. Early works in multichannel protocols gave only passing mention to confidentiality. Smart-Its Friends [10] created a second channel for pairing based on shaking mobile devices together, but the applications envisioned in that work were focused on convenience rather than security. Another early work, the Resurrecting Duckling paper by Stajano and Anderson, stated outright, “We have little to say about confidentiality” [11], recommending standard encryption procedures for the task.

This assumption—that encryption is sufficient to provide the needed confidentiality—carries through much of the subsequent multichannel security research [12]–[16]. These works focus instead on using multiple channels to ensure data authentication. (Entire surveys have been devoted to this specific use [17].) Limited exploration of other security properties began once Wong and Stajano [18] generalized the definition of a “multichannel protocol” to any protocol which establishes a second line of communication in order to strengthen some property of the primary channel. Channel properties such as privacy and directionality have since been considered in addition to authenticity [19], but MICSS is the first real protocol to combine multichannel with secret sharing, rather than simply relying on encryption to ensure confidentiality.

III. Approach

A. Threat Model

MICSS assumes the presence of a strong adversary who has compromised and can modify packets on $n-1$ of the n channels between two hosts. The adversary is assumed to be able to hide his or her presence, so that the compromised channels cannot be identified. Furthermore, since the goal of MICSS is to allow defense in depth when cryptosystems fail, the adversary is also assumed to be able to *break any cryptography* in use on each compromised channel. MICSS guarantees perfect secrecy against this strong $(n-1)$-channel attacker, and a significant challenge in realizing multichannel secret sharing is understanding and defending against such a strong adversary. The design of MICSS—most notably the association protocol described in Section III-D—must address the adversary’s abilities while relying only on the assumption that a single, unidentified channel remains uncompromised.

MICSS is designed to provide *confidentiality* against this adversary. Data authentication has been well studied in previous multichannel research, and message integrity can be achieved by adding hashes to confidential messages. Denial of service attacks are outside the scope of this work.

B. MICSS Architecture

The MICSS protocol is inserted as a bump in the stack between the application and transport layers, as illustrated in Figure 2. This placement allows it to take full advantage of the features of other protocols in the stack. Because MICSS is positioned below the application layer, it coexists seamlessly with application-layer encryption such as TLS, and since it is above the transport layer, it can make use of the semantics and properties of transport-layer protocols.

MICSS uses single-channel TCP as its supporting transport protocol due to several features of TCP—retransmission, flow control, buffering, and in-order delivery—which are exploited to support secret sharing transport (Section III-E). TCP also allows full control over what data is sent on which channel, something not available in existing multichannel protocols such as Multipath TCP [20] and SCTP [21]. These protocols, which are designed primarily to increase throughput and reliability, will retransmit dropped packets on a different channel under certain circumstances. Since this behavior prevents MICSS from guaranteeing that an $(n-1)$-channel attacker will only see $n-1$ of the shares, MICSS cannot be securely implemented on either of these protocols.

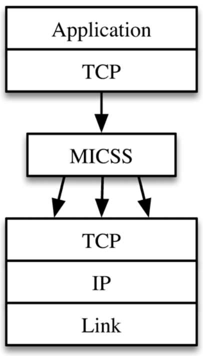


Fig. 2. MICSS as a bump in the Internet protocol stack

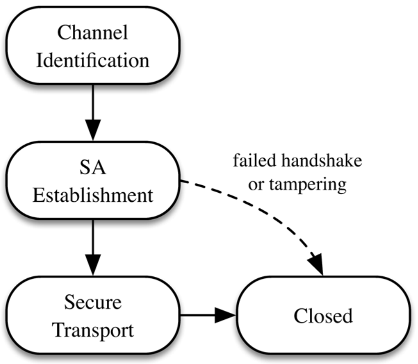


Fig. 3. Phases of a MICSS connection

In order to create a transparent bump in the stack without modifications to the kernel or applications, the iptables routing framework is used to select and intercept outgoing connections. This approach allows for significant flexibility in specifying which connections should be redirected through MICSS. Criteria can be as broad as "every connection destined for port 80 on any host," or as specific as "connections initiated by the user Bob after 10pm on Fridays." Any selected connections are redirected by iptables to the MICSS handler process, listening at port 6522 on the local host. This handler accepts the connection, retrieves the original destination address, and begins the MICSS connection process.

The four phases in the lifetime of a MICSS connection are shown in Figure 3. First, the client must identify all n channels. Second, the client establishes a connection on each of the channels and, together with the server, binds them into a single security association. If this is unsuccessful or tampering is detected, the connection is closed; otherwise, the client and server begin secure transport of data using secret sharing, which continues until the connection is closed by an endpoint.

C. *Channel Identification*

Once a MICSS-capable client intercepts an outgoing connection, its next task is to identify the available channels between itself and the server. Each channel can be specified by the server address to which the client must connect to establish communication on that channel. Only one channel is known at this point: the one over which the client intended to connect before being redirected to the MICSS handler. To obtain this first address, the handler retrieves the original destination of the connection from iptables. Identification of the remaining channels must proceed in such a way that an $(n-1)$-channel attacker cannot interfere, such as by looking them up in a local database of preconfigured addresses or (if the attacker model is temporarily relaxed) by using encryption to augment the initial exchange of addresses.

The first and most straightforward option is to preconfigure server addresses on the client. In this case, MICSS uses the first address obtained from iptables to look up the remaining channels in a database. This is the preferred method to use when possible, because it is local, fast, and safe from any chance of tampering by a network adversary. Even if another method is used initially, the result should be cached in this local database so that subsequent interactions with the server can use the stored addresses.

More options are available if the attacker model is modified, for the duration of a single exchange, to assume that the adversary does not break encryption. If this assumption is allowed, then a shared secret or a public/private keypair for the server may be used to protect the exchange of addresses. Upon connecting to the server's first address, the client generates a nonce and sends it to the server. The server concatenates this nonce with its list of addresses, computes an HMAC or digital signature, and replies to the client with the addresses and this authentication token. The client then verifies the authenticity of the response, caches the address mapping in its local database, and proceeds to the next phase.

D. *Security Association Establishment*

After the client has identified the channels it will use to communicate with the server, it establishes a TCP connection on each channel. The server must now determine, given potentially many independent incoming connections, which of these connections belong to the same client socket. To accomplish this, the MICSS protocol includes an *association handshake*, in which several messages are exchanged after connection establishment to ensure that both the client and the server know which connections belong together. This is a particularly subtle task, due to the adversary's ability to modify packets on an unknown set of $n-1$ channels. The MICSS handshake must recognize the possibility of such an adversary and exploit the existence of the one remaining channel to ensure the adversary cannot exclude that channel from the association without being detected by one or both of the hosts.

The approaches for association establishment found in existing multichannel protocols are not sufficient for MICSS, as they do not defend against an adversary of this kind. Both SCTP and Multipath TCP use only one channel to set up association parameters, leaving them vulnerable to a

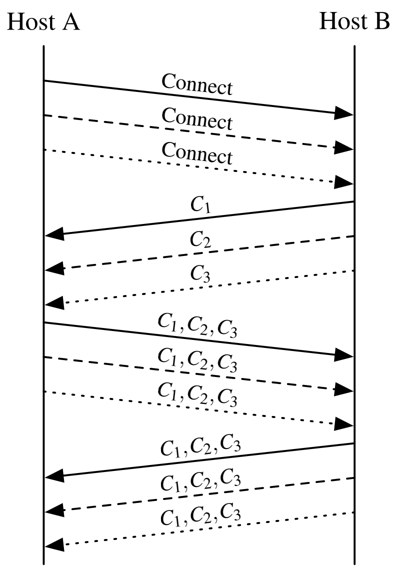


Fig. 4. Client/server security association handshake for n channels (three channels shown, differentiated by line style)

single-channel packet-modifying attacker, let alone an $(n-1)$-channel attacker. In the case of SCTP, the hosts' addresses on other channels are exchanged during association setup. A packet-modifying adversary could manipulate these addresses to redirect data to malicious hosts from channels which would otherwise be uncompromised. Multipath TCP uses the first channel to exchange random keys, the knowledge of which allows later connections to add themselves to the association. This also could be eavesdropped and manipulated to compromise the process (a recognized attack against Multipath TCP [22]).

MICSS, therefore, has a unique security association handshake, shown in Figure 4:

1) Host A simultaneously establishes a TCP connection to Host B on each channel.
2) Host B, which does not yet know that these connections are related, chooses a unique "cookie" C_i for each individual incoming connection and sends it back on that channel.
3) Once the client has received a cookie on every channel, it collects them into a single list and sends this entire list to the server on each channel.
4) If the server receives the same set of n cookies from n connections, it establishes an association between all of them and sends the full list of cookies to the client on each of those channels.
5) If the client receives the correct response on every channel, it continues to the secure transport phase.

This handshake sends an entire set of cookies from client to server on every channel, then back from server to client in the same fashion. This guarantees that the uncompromised channel is used twice: once to convey the client's idea of which connections are associated to the server, and once to convey the server's idea of the same back to the client. If either host receives, on any channel, something other than the list of cookies it expects to see, it immediately aborts the association and closes the connections. Upon successful completion of this handshake, the hosts begin transmitting data.

E. Secure Transport

Once the multichannel security association has been established, MICSS begins its data transport phase. In the current iteration of the protocol, any n-out-of-n secret sharing scheme can be used to safeguard message data over n channels. Such a scheme will take a message M as input and create shares $S_1, \ldots, S_n$, with all n shares being necessary to reconstruct M. Figure 1 illustrates how this interacts with the other layers in the MICSS protocol. When an application sends a message, the secret sharing layer demultiplexes it into n shares. These shares are passed to the multichannel layer, which transmits each share S_i over channel i. On the receiving host, the multichannel layer reads the shares from each channel and passes them to the secret sharing layer, which multiplexes them to reconstruct the original message.

The specific n-out-of-n secret sharing scheme used in the MICSS reference implementation is the XOR one-time pad. XOR is a natural fit given its efficiency and simplicity. Using this scheme, the sender generates random pads $S_1, \ldots, S_{n-1}$ as the first $n-1$ shares, each of which is the same length as M. The remaining share, S_n, is determined as follows:

$$S_n = S_1 \oplus \cdots \oplus S_{n-1} \oplus M.$$

In order for the MICSS receiver to reconstruct a network packet, it must receive all of the shares, and it must identify that these shares correspond to each other. The former requirement is satisfied by the reliability of the underlying TCP streams: if any share is dropped, its retransmission will be handled automatically by TCP. MICSS fulfills the latter requirement by manipulating TCP's flow control to synchronize the sliding windows of all of the channels. Its delivery algorithm ensures that the channel which is furthest behind is always read first, forcing faster channels to wait even if they have new data available. As these channels' TCP buffers begin to fill, their sliding window sizes will be reduced until all of the channels are transmitting data at approximately the same rate. Then, due to in-order delivery guarantees, the next bytes to be read on each channel will correspond to the next data to be reassembled.

Finally, when all corresponding shares of a message M have been received, MICSS reconstructs it as follows:

$$M = S_1 \oplus \cdots \oplus S_{n-1} \oplus S_n$$

and delivers the resulting message to the destination.

IV. Performance Evaluation

The MICSS protocol guarantees perfect secrecy against an $(n-1)$-channel attacker, and it does so with reasonable performance. The first part of this claim follows directly from

TABLE I
RESULTS OF NETWORK MICROBENCHMARKS

Channel	A	B	C	MICSS
Throughput (Mbps)	14.9	9100	9100	14.9
Latency, mean (μs)	8600	13.9	13.9	7900
Latency, std. dev. (μs)	12000	108	108	43000

Shannon's one-time pad proof [6], and the second part is validated by collecting a number of micro- and macrobenchmarks for a reference implementation of the protocol. The setup used in these experiments is a pair of Dell PowerEdge M620 blades with dual quad-core processors at 2.4 GHz, running Linux 3.14.1. The hosts are connected by three channels. Channel A is a point-to-point wireless Ethernet link between two Tenda W311MI USB interfaces at close range, and channels B and C are both quiescent wired Ethernet networks. The address of the server on each channel is configured at the client prior to the experiment.

A. Network Microbenchmarks

The first step in the performance evaluation of MICSS is a comparison with single-channel TCP in terms of throughput and latency. This is accomplished by collecting microbenchmarks with the Netperf utility [23], which measures both throughput and latency for a TCP stream. The benchmark is first run directly using TCP over each individual channel, and then run using MICSS over all three channels. The results of these benchmarks are summarized in Table I and discussed below.

The theoretical maximum performance of a protocol with the information-theoretic security guarantees of MICSS is dictated by the performance of the individual channels. To ensure perfect secrecy against an $(n-1)$-channel attacker, the share being transmitted on each channel must be as long as the original message itself. Since all correlated shares must reach the destination host before a message can be delivered, the slowest channel will determine the maximum throughput of the overall multichannel system. In the Netperf benchmarks of MICSS, the throughput of each wired channel was 9100 Mbps, and that of the wireless channel was 14.9 Mbps. MICSS is able to operate at the theoretical maximum line speed of 14.9 Mbps. In terms of latency, it achieved slightly lower average latency, although with a noticeably higher standard deviation.

MICSS also degrades gracefully in the presence of lossy channels. To demonstrate this, the same Netperf benchmark is executed, this time with each network interface configured to drop packets at random, with probabilities ranging from 0% to 15%. The results show that MICSS exhibits graceful degradation for lossy channels; Figure 5 displays its performance alongside that of the limiting wireless channel. Since all three channels are lossy, MICSS must deal with three times as many losses as single-channel TCP, so it begins to degrade approximately 2% sooner than TCP over the wireless link alone. (In experiments where only the wireless link was lossy, the results for MICSS showed no significant difference from single-channel TCP.)

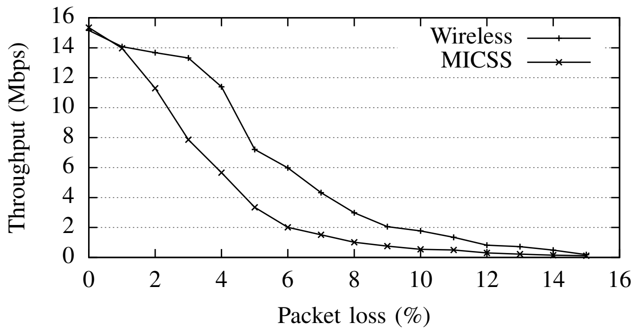
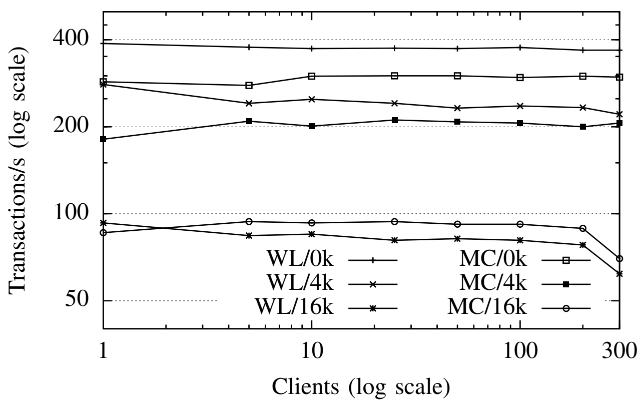


Fig. 5. Graceful degradation of MICSS over three lossy channels



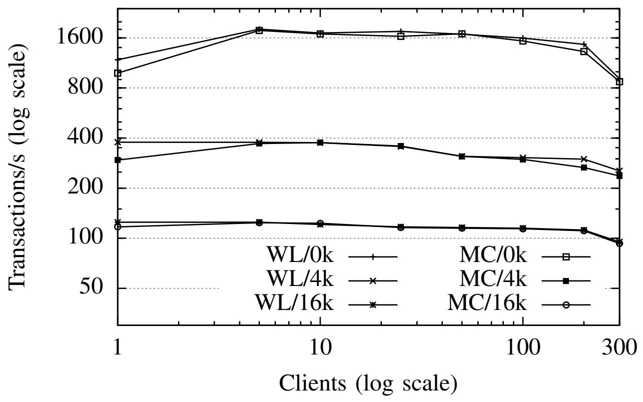


Fig. 6. Concurrent HTTP requests for 0k/4k/16k files. Top: one request per connection. Bottom: multiple requests per connection using Keep-Alive.

B. Network Macrobenchmarks

MICSS also performs well in large-scale network applications such as Web servers. To demonstrate this, an Apache server was added to the experimental setup, and the `weighttp` utility [24] was used to simulate a large number of clients concurrently issuing HTTP requests to the server. This utility measures the number of HTTP transactions the server can successfully complete per second, given a specific number of

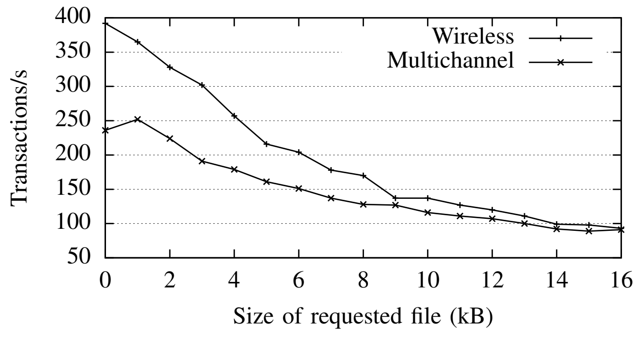


Fig. 7. Amortizing MICSS association overhead in HTTP requests

concurrent clients. Three files of different sizes were served: one empty, one four kilobytes, and one sixteen kilobytes.

The results of `weighttp` from 1 to 300 clients are shown in Figure 6. In the first experiment, clients create a new connection for every request, and the overhead from the MICSS association handshake is visible. In the second experiment, HTTP Keep-Alive is enabled, allowing clients to amortize the connection overhead by keeping their connections open as they make requests. In none of the tests is there a significant degradation in MICSS performance without a corresponding drop in the control, indicating that the limiting factor beyond 200 clients is the wireless channel itself and not MICSS.

The MICSS association overhead is also noticeable in the fact that the initial overhead for the 16-kilobyte file is much less than that of the empty file. To illustrate this more clearly, one more benchmark is collected, in which clients make a single request per connection and vary the size of the requested file from empty to 16 kB. The results (Figure 7) show that the total overhead is significant when requesting an empty file, since the response time is almost completely dominated by connection setup. As the size of the file increases, the initial overhead is amortized over the entire connection, and the performance converges with that of single-channel TCP.

V. Conclusion and Future Work

In its current form, MICSS is tailored to provide stream semantics with perfect secrecy against an $(n-1)$-channel attacker. Future work will extend the protocol to other uses by modifying this approach in three ways. First, re-engineering the protocol to operate at the network layer will enable support for best effort semantics as well as data streams. Second, the new semantics will allow the use of a variety of other secret sharing schemes—for example, (k,n)-threshold schemes, which provide a parameter k that can be adjusted to trade a degree of information-theoretical security for increased performance. Third, integrity protection may then be added to the protocol by prepending each message with a hash of its contents before secret sharing is applied. In the current protocol this would require the addition of framing, but a re-engineered version would be able to exploit the framing which is already present at the network layer.

This paper has presented MICSS, the first realization of a multichannel secret sharing protocol. It ensures perfect secrecy against an $(n-1)$-channel attacker, and it is completely transparent to applications, networks, and the kernel. Cryptosystems on their own only provide a single line of defense, but when cryptosystem implementations fail, MICSS provides an effective, practical, and flexible means to defend in depth for confidential network communications.

References

[1] R. J. Anderson, "Why cryptosystems fail," in *Proc. ACM CCS*, 1993.
[2] Codenomicon, "Heartbleed bug," http://heartbleed.com/, Apr. 2014.
[3] A. Shamir, "How to share a secret," *CACM*, vol. 22, no. 11, 1979.
[4] G. R. Blakley, "Safeguarding cryptographic keys," in *Proc. National Computer Conference*. IEEE Computer Society, 1979, pp. 313–317.
[5] ——, "One time pads are key safeguarding schemes, not cryptosystems," in *1980 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1980, pp. 108–113.
[6] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.
[7] H. Yamamoto, "On secret sharing communication systems with two or three channels," *Information Theory, IEEE Transactions on*, vol. 32, no. 3, pp. 387–393, 1986.
[8] W. Lou and Y. Fang, "A multipath routing approach for secure data delivery," in *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, vol. 2. IEEE, 2001, pp. 1467–1473.
[9] R. A. Vasudevan and S. Sanyal, "A novel multipath approach to security in mobile ad hoc networks (MANETs)," *arXiv:1112.2128*, 2011.
[10] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H.-W. Gellersen, "Smart-Its friends: A technique for users to easily establish connections between smart artefacts," in *Ubicomp*, 2001, pp. 116–122.
[11] F. Stajano and R. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks," in *Security Protocols*. Springer, 2000.
[12] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong, "Talking to strangers: Authentication in ad-hoc wireless networks," in *NDSS*, 2002.
[13] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication," in *Security and Privacy, 2005 IEEE symposium on*. IEEE, 2005, pp. 110–124.
[14] F.-L. Wong and F. Stajano, "Multichannel security protocols," *Pervasive Computing, IEEE*, vol. 6, no. 4, pp. 31–39, 2007.
[15] C. Gehrmann and K. Nyberg, "Enhancements to Bluetooth baseband security," in *Proceedings of Nordsec*, vol. 2001, 2001, pp. 191–230.
[16] M. Cagalj, S. Capkun, and J.-P. Hubaux, "Key agreement in peer-to-peer wireless networks," *Proc. IEEE*, vol. 94, no. 2, pp. 467–478, 2006.
[17] L. H. Nguyen and A. W. Roscoe, "Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey," *Journal of Computer Security*, vol. 19, no. 1, pp. 139–201, 2011. [Online]. Available: http://iospress.metapress.com/content/y12221481181x146/?p=596ef23449cc4e8a91f6c4a7a3018fff&amp;pi=4
[18] F.-L. Wong and F. Stajano, "Multi-channel protocols," in *Security Protocols Workshop*, 2005, pp. 112–127.
[19] J.-H. Hoepman, "The ephemeral pairing problem," in *Financial Cryptography*. Springer, 2004, pp. 212–226.
[20] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," RFC 6824 (Experimental), Internet Engineering Task Force, Jan. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6824.txt
[21] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 6096, 6335, 7053. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt
[22] M. Bagnulo, "Threat analysis for TCP extensions for multipath operation with multiple addresses," RFC 6181 (Informational), Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6181.txt
[23] R. Jones, "The Netperf homepage," http://www.netperf.org/.
[24] Lighty Labs, "weighttp," http://weighttp.lighttpd.net/wiki.