Secure Communication: a New Application for Active Networks

Manuel Günter, Marc Brogle and Torsten Braun

Institute of Computer Science and Applied Mathematics (IAM),
University of Berne, Neubrückstrasse 10, CH-3012 Bern, Switzerland
`http://www.iam.unibe.ch/~rvs/`

Abstract. SplitPath is a new application for the easy, well-known and provably secure one-time pad encryption scheme. Two problems hinder the one-time pad scheme from being applied in the area of secure data communication: the random generation and the distribution of this random data. SplitPath exploits the flexibility of code mobility in active networks to address these problems. Especially the random generation is studied in more detail.

1 Introduction

A wide variety of encryption algorithms is in daily use to protect data communications. However, none of these algorithms is proven to be secure. A well-known algorithm exists which is very simple and perfectly secure: the one-time pad [Sch96]. It works as follows: Assume you want to encrypt a bit string P of n bits ($P, p_i \in \{0,1\}, i \in 1..n$). For that purpose you take a string of equally distributed and independent random bits $R, r_i \in \{0,1\}, i \in 1..n$ and xor (addition modulo 2) it bit-wise with P, resulting in the ciphertext $C, c_i = r_i \oplus p_i$. To decrypt C, R is xor-ed to C again. This works because $r_i \oplus (r_i \oplus p_i) = p_i$. R must be destroyed after the decryption. It is assumed that once R is generated, it is used for encryption and decryption *only once*. That's why the scheme is called one-time pad. Under these assumptions the algorithm is provably secure. It is impossible to gain any knowledge of P without knowing R, because *every possible plaintext* could have lead to a given ciphertext. Furthermore, in contrary to commercial encryption algorithms, the one-time pad needs only one very light-weight operation (xor) for encryption and decryption. However, the one-time pad is not practical for secure data communication for the following reasons:

– **Lack of random bits.** The one-time pad needs a irreproducible random bit-stream (R) of the same length as the message.
– **Ensuring single usage.** The receiver and the sender must both exclusively possess the same pad (R). How can the pad securely be established by the communication partners?

This paper presents an approach using active networking [CBZS98,TSS$^+$97] to address both problems. An active network consists of active (programmable) network nodes. The data packets that are transmitted through an active network can contain code that the active nodes execute. Active networking is an instance of the mobile agents paradigm tailored to networking needs. Active network packets (also called capsules) access the networking functionalities of the nodes (e.g. forwarding and routing) and change these functionalities for packets or classes of packets.

This paper presents how (for a given application scenario) active networking enables us to use the one-time pad with its provable security and lightness for secure data communication.

In section 2 we present the basic idea how to address the distribution of the random (problem 2). Section 3 describes how to generate the necessary random (problem 1) and what the pitfalls are. An existing implementation using the well-known active networking tool ANTS [WGT98] is presented in section 4. Section 5 presents performance measurements and section 6 concludes the paper.

2 Distribution of Keys and Data

We said that with enough good random bits available, we can create an uncrackable bit stream using the one-time pad. However, the receiver must also possess the random bits. A straight-forward solution is to first deliver the random bits in a secure manner and later transmit the data. The sender could, for example, hand a magnetic storage tape to the receiver, containing the random bits. This is a secure but not very flexible application of the one-time pad. The use of a secure communication medium[1] allows the communicating parties to communicate later using an insecure communication medium. But even if the medium for sending the random is not secure, the scheme still works as long as no attacker has access to *both* random and message bits. This principle is e.g. used when encryption keys for data communication are exchanged using the postal service or the telephone system. The SplitPath idea goes one step further. The random bits (interpreted as the key) and the cipher text bits (plaintext xor-ed with the random bits) are sent along at the same time on the same media but *on different paths*. In general this scenario is not secure any more, since an attacker can eavesdrop both the random string and the encrypted message and thus easily decrypt the original message. In a network with centralised management at least the network provider will always be able to do this. However, if the network is partitioned in autonomous sub-networks (domains), as for example the Internet is, and if the two paths are entirely in different domains, an attacker will have significantly more trouble. Thus, the application of SplitPath requires the following preconditions:

- SplitPath traffic enters the untrusted networks only in split form (either random bits or xor-bits).
- The paths of corresponding random bits and xor-bits never share the same distrusted network.
- The distrusted networks do not trust each other.

These preconditions limit the application scenario of SplitPath, but there are cases where the prerequisites are met.

[1] Assuming that the physical delivery of the magnetic tape is secure.

- **Multi-homing.** Today's Internet service providers (ISP) compete with each other. Most of the larger ones own their own physical network infrastructure. It is not uncommon that customer networks are connected to more than one ISP.
- **Heterogeneous and redundant network technology.** A modern global data network like the Internet consists of all kinds of different network technologies (e.g. optical networks and satellite links) running different link layer protocols (ATM, Frame Relay etc.). These link layer networks are usually inter-connected on many redundant links.
- **International tension.** Unfortunately, many neighbouring countries tend to be suspicious about each other. The geographical location of-, and the relation between some nations can provide an ideal application scenario for the SplitPath scheme.

The generic situation is depicted in figure 1. At the split point the data packet is split into a random packet (the pad) and the xor result of data and pad. The resulting packets are sent along disjunct network paths. When they arrive in the trusted receiver network (at the merge point) then they are merged back into the original data packet.

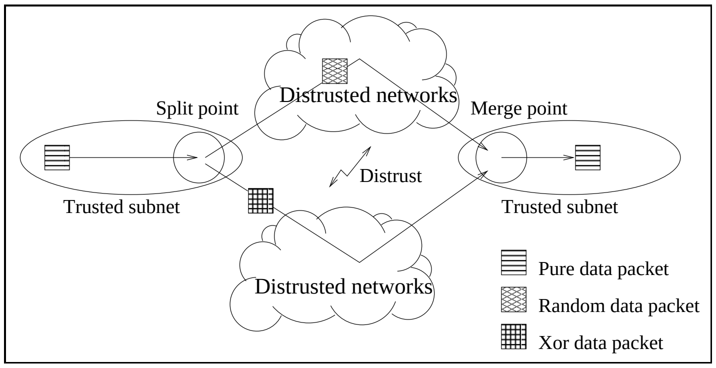

Fig. 1. The application scenario for SplitPath.

We distinguish between the sender, the receiver, a split point and a merge-point. Obviously, the split- and merge point needs to be located on a trustworthy site. Depending on the sender, receiver and the network topology the ideal location of these points varies, thus their implementation cannot be preconfigured in few network nodes. Active networking brings the flexibility we need here. With active networking the data packets can dynamically setup their private split (merge) functionality in the nodes which are appropriate for them. No router configurations are necessary. SplitPath capsules contain code that dynamically implements the split and merge capabilities and controls the routing of the packets. These mechanisms are described in section 4.

The next section shows how active networking can also help us getting the 'good' random bits which are needed in the split point.

3 Generating Random

For SplitPath, like for most crypto-systems, the availability of high quality random material is crucial. The random bits must be independent and equally distributed in order to be unpredictable. However, it is not easy to acquire such random in a computing environment. Irreproducible random values can solely be created by real world interaction. Examples are: mechanical random devices (e.g. lottery machines, dices), physical random (e.g. radioactive decay) and human behaviour (e.g. keyboard interrupt times). Multiprocessing and networking devices can also be a source of random bits [Wob98].

In SplitPath, we propose to use unpredictable traffic characteristics as seen in networking devices and generate random bits with them. Active networking allows the capsule to use its *own* varying queueing times within the network. The capsule, being autonomous, can keep information about its creation time and ask the nodes on its way about the local time. Note, that clock skew is actually good for the random generation because it introduces another factor of incertitude.

The idea is that by travelling through the net each capsule generates random bits for its own use. However, e.g. an IP packet can contain 64 KBytes of data. It needs the same amount of random bits for applying the one-time pad.

3.1 Quantity and Quality of SplitPath Random

The quantity of random bits gained by network performance measurement is limited. This is a problem but we have many options to cope with the situation:

1) Limit the payload of the capsule. Programmable capsules can fragment themselves to smaller payload sizes. This produces more packets and thus more random data per payload byte. It also adds bandwidth overhead. Note however, that congestion eases the production of random bits, since it involves a lot of non-predictable behaviour (see also section 5).

2) Generate more random bits by sending empty packets. When capsules can store data in nodes (see section 4) then we can use the idle time to send empty capsules that store their performance statistics in the split-node. Later, when payload is transported, the capsule can use the stored information to generate random.

3) Multi-hop random generation. If the capsule executes at several nodes before the split node, it can calculate its performance statistics there, too. The capsule takes the gained random bits with it and uses them at the split node. Care must be taken when distilling random bits from several nodes, because they are probably not completely independent.

These options do not affect the strength of the one-time pad, but they limit the effective throughput of data. Another approach is to use a pseudo-random function [Sch96]. This function uses the collected random data as a seed and generates a bit sequence of arbitrary length that can be used to xor the data. However, the next paragraph explains why such 'stretching' of the random bits affects the perfect security of the one-time pad.

When an attacker tries to decrypt an unknown ciphertext the most straight-forward thing to do is to decrypt the message with every key and see if the result looks like a meaningful message. This is also called a *brute-force attack*. The one-time pad is immune against such attacks, because in his search through the key space the attacker will find *every* meaningful message that can be encoded in that length. However, once we start expanding the random key string by using it as a seed to a pseudo-random function, we will loose this nice property of the one-time pad. Based on Shannon's communication theory of secrecy systems, Hellman [Hel77] showed that the expected number P of different keys that will decipher a ciphertext message to some intelligible plaintext of length n (in the same language as the original plaintext) is given by the following formula: $P = 2^{H(K)-nD} - 1$. P indicates the success probability of the brute-force attack. If P is small then the brute-force attack will deliver only few possible plaintexts, ideally only one, which then can be assumed to be the original message. $H(K)$ is the entropy of the crypto-system used and D is the entropy of the encoded language. Using this formula we can show [GBB00] that for ASCII encoded English text each payload byte should be protected by at least 7 random bits. This is a very small stretching factor of $8/7$.

Random expansion in practice. When we want to 'stretch' the random data by larger factors we can not rely any more on the perfect secrecy of the one-time pad. Instead, we have to carefully design the pseudo-random generator. First of all, the seed length must be large. We propose 128 bits. While the previous paragraph showed that a brute attack in principle will lead to the decryption of the packets (in the worst case there is only one random bit per packet), in practice the attack will not be successful, because there are too many bit combinations to try (an average of 2^{127}). If a million computers would each apply a billion decryption tries per second the average search would still last about 5*10^{15} years.

Second, the pseudo random generator should resist cryptanalysis. Many such generators exist and are used for so-called stream ciphers [Sch96]. Using SplitPath with expanded random is very similar to using a stream cipher in that both xor the plaintext with a secure pseudo random bit stream. However, SplitPath differs from stream ciphers in that it uses the pseudo random generator only at the sender side. Furthermore, the flexibility of an active network platform allows SplitPath to dynamically change the generator used, even during an ongoing communication. Finally, the seed of the generator is updated frequently (as soon as enough random bits have been collected by the capsules), and the seed is random. This is different from e.g. the stream cipher A5 (used for mobile telephony) which uses a preconfigured seed.

4 Implementing SplitPath in an Active Network

4.1 Implementing SplitPath with the Active Node Transfer System ANTS

We implemented the SplitPath application using the active node transfer system ANTS [WGT98]. ANTS is a Java based toolkit for setting up active networking testbeds. ANTS defines active nodes, which are Java programs possibly running on different machines. The nodes execute ANTS capsules and forward them over TCP/IP. ANTS defines a Java class `Capsule`. The class contains the method `evaluate` which is called each time the capsule arrives at a node. New capsule classes can implement new behaviour by overriding the `evaluate` method. The node offers services to the capsule such as the local time, forwarding of the capsule and a private soft-state object store (called *node cache*). Collaborating capsules can be grouped to protocols. Capsules of the same protocol can leave messages for each other using the node cache. We defined such a protocol to implement the SplitPath concept as presented in section 2 by introducing three new capsule subclasses.

1) The `Pathfinder` capsule marks nodes as splitting or merging points and sets up the split paths using the node caches. Note that several split and merge points can be set up per communication. Currently, `Pathfinder`s are parametrised and sent by applications. We foresee to implement them with autonomous intelligence.

2) The `Normal` capsule is the plaintext message carrier. It checks if it is on a splitting point. If not, it normally forwards itself towards the destination. If it is on a splitting point, it applies the one-time pad computation to its payload. This results in two `Splitted` capsules, one carrying the random bits, the other the xor-ed data in the payload. The `Normal` capsule tells the node to forward the two newly produced `Splitted` capsules instead of itself, thereby using the information that was setup by the `Pathfinder`.

3) The `Splitted` capsule carries the encrypted- or the random data along a separate path. It forwards itself using the information that was setup by the `Pathfinder`. It checks if it has arrived on a merge point. If so, it checks if its split twin has already arrived. In that case it xor-s their contents (decryption) and creates a `Normal` capsule out of the result. If the twin is not there, it stores itself on the node cache to wait for it.

Applications and Interfaces. We wrote two applications that send and interact with the implemented capsules. The first application provides a graphical interface which allows the user to dynamically set up split and merge points using the `Pathfinder`. Furthermore, the user can enter and send text data using the `Normal` capsule. We also extended the nodes with a sniffing functionality. Such 'spy nodes' log capsule data to validate and visualise the SplitPath encryption. We implemented a second application which transfers a file (optionally several times) in order to test the performance by generating load.

4.2 Random Generation and the Application of the One-time pad

In order to be able to send large packets, we decided to use pseudo random generators to extend the collected random bits (see section 3). Each `Normal` capsule contains its creation time. When arriving at a split node, it uses this time and the node's current time to calculate the delay it has so far. It stores the last few bits as random seed. The number of bits is configurable and depends on the clock resolution. Unfortunately, the Java system clock resolution used by ANTS is bound to one millisecond and the delay is in the order of few milliseconds. Therefore, we used only the least significant bit. Thus, every packet stores one random bit in the split node (using the node cache). This bit can be considered as random, since it is influenced by the speed and current usage of computing and networking resources. As said before, the chosen seed length is 128 bits. So for every 128th capsule a complete seed is available. This capsule uses the seed to store a new random generator in the node cache. The next capsules use the generator for their encryption, until enough fresh random has been collected to install a generator with the new seed (key refreshing). For the bootstrapping we foresee two schemes. Either 128 empty packets are sent, or a (less secure) seed is used that can be generated by the first packet.

We have implemented two random generators. The first one is based on the secure one-way hash function MD5[2]. The seed is stored in a byte array of 16 bytes. Furthermore, there is an output buffer of 16 bytes containing the MD5 hash value of the seed buffer. The generator delivers 8 bytes of the output buffer as pseudo-random values. Then, the seed is transformed and the output buffer is updated (MD5 hash). The 'one-way' property of MD5 assures that an attacker cannot reconstruct the seed. Thanks to the avalanche property of MD5 the transformed seed produces an entirely different hash. Our seed transformation is equivalent to the increment of a long integer (8 bytes) by one. Thus, the seed only repeats after 2^{64} transformations. Long before that, SplitPath will replace the seed with freshly gathered random values. We think that for the presented reasons this pseudo random generator is reasonably secure. Nevertheless, we implemented also the pseudo random generator of RC4 as an alternative. RC4 is a stream cipher developed by RSADSI. It is used in applications of e.g. Lotus, Oracle and Netscape (for details see [Wob98]).

5 Evaluation of SplitPath

In order to evaluate SplitPath we ran the implementation on our institute network. Six ANTS nodes were set up on six different machines (sender, receiver, a split and a merge node and two sniffers; see figure 2). The split node ran on a SPARCstation 5/170. The encrypted capsules ran over two different subnets and were merged in a machine of a third subnet. The subnets are 100 Mbps Ethernets. We used the aforementioned file transfer application to generate load. Our interest was focussed on the quality of the encryption. We measured this by collecting all generated seeds and apply statistical tests. Furthermore, we applied statistical tests to the MD5 pseudo random generator presented in the previous section.

In order to test the quality of the MD5 random generator, we initialised it with all seed bytes set to zero. Then we fed its output into a framework for statistical tests. Testing with samples of 40 MByte size, the produced data succeeded the byte frequency test, the run test [Knu81] and the Anderson-Darling test [IETF RFC 2330]. This is no prove that the generated pseudo-random bits are of high quality, but it shows that they are not flawed.

Seed generation. We evaluated the generated seed bytes using statistical tests. For example we analysed 3K seed bytes (192 complete seeds, protecting 24576 packets). The seeds pass the byte frequency test (χ^2 test on the distribution of the measured byte values [Knu81]). Unfortunately, in few cases we also experienced seed generation that

[2] ANTS includes the Message Digest 5 (MD5) [IETF RFC 1321] functionality.

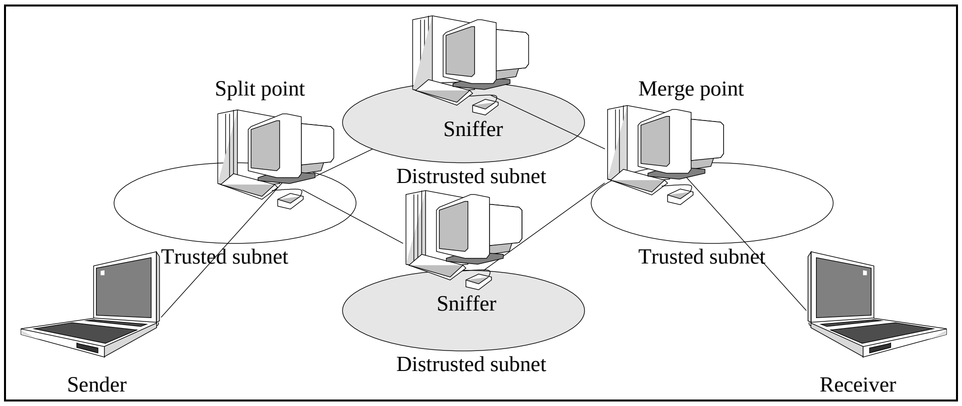

Fig. 2. The network topology for the evaluation.

was not uniformly distributed. There we see some concentrations of byte values especially around the value 0. Our investigation revealed three reasons that come together to form this effect. (1) The coarse resolution of Java's clock. (2) ANTS does not send one capsule per packet, and the packet code is not send within the capsule, but dynamically loaded and cached. Thus, consecutive capsules are handled immediately after each other without delaying I/O operations. (3) The local network used has very low delay and jitter. These problems are not discouraging because normally they do not all come together and there are also countermeasures. By introducing congestion we can show that given realistic wide area delays as studied e.g. for the Internet [EM99], the seeds are equally distributed. Figures 5 and 5 show the seeds of two samples. Each sample shows the single byte values (2-complement) in the same order as they were created. Figure 5 represents a sample suffering of the previously mentioned problems, figure 5 represents a corrected sample.

There are many more options to improve the seed quality. Additional use of other random sources for example. We foresee the exploitation of additional sources offered by active networking e.g. execution times of capsules or properties of the node cache. Finally, we can once again exploit the fact that the capsules are programmable. The procedure for collecting seed values can easily be extended to contain statistical methods to test the seed before it is used. So if for some unforeseen reason the seed is not good enough, the capsule uses the old random generator a little longer until more random bits are collected. Also, methods described in [Sch96] can be used by the capsule to distill random from biased random bits.

6 Conclusion

We presented SplitPath, a new application for the well-known and provably secure one-time pad encryption scheme. SplitPath uses the ability of active networks and the trust relations in heterogeneous networks to solve the two problems which otherwise render the one-time pad scheme useless: the random generation and the distribution of this

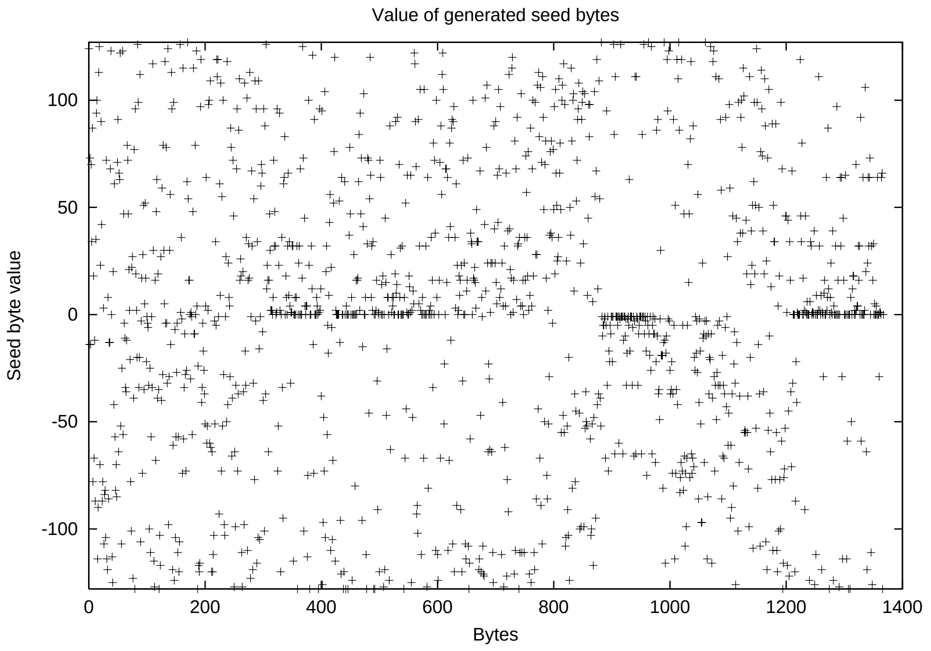

Fig. 3. Biased seed.

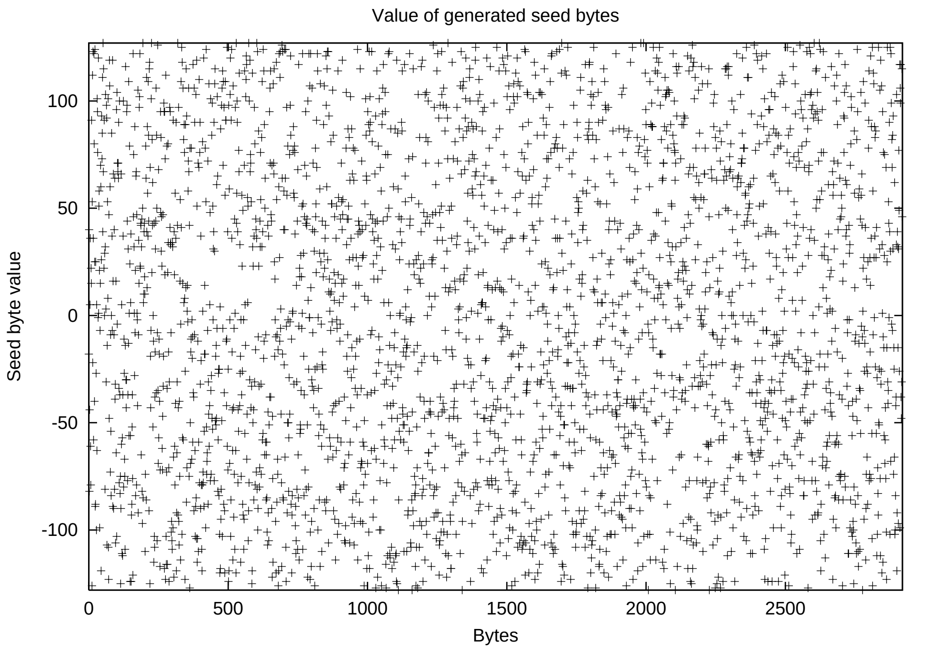

Fig. 4. Good seed.

random data. SplitPath can dynamically set up and use disjunct paths through network domains that do not collaborate, such as competitive network providers or countries. One-time pad encrypted data and random data is forwarded separately on these paths. Active networking allows SplitPath to encrypt and decrypt at any trusted node. Active networking not only allows SplitPath to dynamically set up the one-time pad splitting inside of the network, it also helps to collect good random. This can be very useful for other crypto-systems, too. SplitPath implements data capsules that use the network delays that they experience as initialisation for the random generation. With SplitPath we present an application in a (albeit specific) environment which cannot be implemented using conventional 'passive' networking, thus we promote the future study and (hopefully) deployment of active networking.

References

[CBZS98] K. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz. Directions in active networks. *IEEE Communications*, 36(10), October 1998.

[EM99] Tamas Elteto and Sandor Molnar. On the distribution of round-trip delays in TCP/IP Networks. In *Proceedings of the 24th Conference on Local Computer Networks LCN'99*, pages p.172–181. IEEE Computer Society, October 1999.

[GBB00] M. Günter, M. Brogle, and T. Braun. Secure communication with active networks. Technical Report IAM-00-007, IAM, 2000. www.iam.unibe.ch/∼rvs/publications/.

[Hel77] M. E. Hellman. An extension to the shannon theory approach to cryptography. *IEEE Transactions on Information Theory*, IT-23(3):p. 289–294, May 1977.

[Knu81] D. E. Knuth. *The art of computer programming*, volume 2 Seminumerical Algorithms. Addison-Wesley, 2 edition, 1981.

[Sch96] B. Schneier. *Applied Cryptography*. John Wiley and Son, 1996.

[TSS$^+$97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[WGT98] D. Wetherall, J. Guttag, and D. L. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH '98*, April 1998. San Francisco.

[Wob98] Reinhard Wobst. *Abenteuer Kryptologie*. Addison-Wesley, 1998.