Secure Streaming Media and Digital Rights Management

Deepali Holankar
Department of Computer Science
San Jose State University
San Jose, CA 95192
dbrahmbhatt@pacbell.net

Mark Stamp
Department of Computer Science
San Jose State University
San Jose, CA 95192
stamp@cs.sjsu.edu

Abstract

When dealing with streaming media, integrity protection and replay prevention are difficult security challenges. In this paper we discuss the design and implementation of a secure streaming media system. Our approach incorporates ideas from the field of digital rights management (DRM).

1 Introduction

Due to recent advances in technology, streaming media—including real time audio and video conferencing—has become a popular and important aspect of telecommunication. Often, streaming content providers do not want the end users to capture and duplicate the streamed media, since once such data is captured it can be freely re-distributed without any control from the source. In a similar vein, privacy and integrity are crucial for successful video or audio conferencing. Note that data encryption [2] does not generally suffice, since it leaves the system vulnerable to duplication and recording after decryption.

Possible sources of streamed media include stored data, live broadcasts and interactive conferencing. We focuses on live broadcast media as our case study but the results are easily extended to stored media or interactive conferencing.

Secure real-time communication over insecure networks generally involves two major security concern—authentication and privacy [5]. The real-time transport protocol (RTP) is used to transport a media stream between two multimedia terminals [6]. Secure RTP employs encryption techniques on the RTP stream [7]. In this paper, we argue that applying some concepts of digital rights management (DRM) can help to provide integrity protection and replay protection within reasonable bounds. We discuss the specific security features borrowed from the field of DRM and how best to incorporate these into a multimedia application used for transmitting, receiving and processing streaming media over the Internet. We have implemented a prototype of our secure streaming media system and we give some performance results.

2 Digital rights management

Digital Rights Management (DRM) attempts to provide for the secure delivery of digital content with restrictions on the usage of the content after delivery. For example, the provider of a piece of digital content might want to restrict the end-user's ability to duplicate the information. Such restrictions are necessary if the provider is to maintain any control on the distribution of the content. In contrast to classic cryptography—which aims to protect against an unintended recipient—the protection provided by a DRM system is primarily aimed at the legitimate recipient. When seen in this light, it is clear that cryptography is only a very small part of a DRM solution.

In addition, DRM protection must stay with the content even after delivery. In the DRM literature, this required level of protection—which goes beyond the protection that standard cryptography can provide—is often referred to as "persistent protection". For further background information on DRM, including an outline of a complete DRM system, see the article [8].

Consider a scenario in which company A wants to stream a live baseball game to N clients. Company A does not want its evil competitor, company B, to hack their media stream and add noise or distortion to the signal. Moreover Company A does not want any of its clients to record the game and redistribute it. Company A only wants to allow paying customers to have access to the media stream. Digital rights management is designed to deals with such issues.

Below, we proposes a model for secure streaming media that employs some features commonly used by digital rights management systems. In the streaming media scenario, these features primarily provide replay protection, which is lacking in current approaches to streaming media delivery.

Of course, any media streamed to a personal computer can be recorded using an analog device. For example, the video displayed on the monitor screen can be captured via screen shots. In the DRM world, this fundamental problem is known as the "analog hole" [1] and is considered beyond the boundary of protection provided by a DRM system. The DRM philosophy is to make an attack on the system as difficult as possible, while realizing that perfect protection cannot be achieved in the current computing environment.

Our proposed model also does not deal with the storing of streamed data on hard disk or other such media. The security issues concerning storing of copyright material and its reproducibility are separate issues that are not the concern of this paper.

In addition, this paper does not contain a detailed discussion of key management issues. Key management techniques are well established; see, for example, [5] for further information.

What this paper does provide is a proposed techniques to achieve a measure of replay protection and message integrity for streamed media. Of course, it is possible to garble or destroy the encrypted media stream, thus rendering it useless to the legitimate recipient. This paper does not discuss specific protection against such malicious attacks.

3 Assumed background

We assume that the reader has some background knowledge in the areas of security engineering, network security protocols, cryptography and authentication. Good sources for such information include [5] and [9].

The crucial point that we want to emphasize is that a system can only be as secure as its weakest link. In practice, systems are often constructed employing strong and secure cryptographic techniques, yet the result is an insecure system. Therefore, we must analyze our design from end-to-end in order to determine whether there are any security weakness.

4 Proposed model

Digital rights management is an evolving field. The current DRM market includes many proprietary systems as well as open source solutions such as [4]. However, the majority of DRM systems focus on licensing management and rights. Though license management constituents an important part of a DRM, it cannot yield a secure system by itself. DRM can only be successful if there exists a complete security chain that begins with the data transmission and persists beyond the point where the client accesses the content. We have incorporated certain DRM features into our secure streaming media system. These features include license management, also several additional security features, which are discussed below.

This paper is focused on enhancing the rights enforcement ability for streaming media on the client. Of course, it is crucial that the entire system be secure in order to avoid a weak link in the process. Therefore, we give an overview of the entire system, but focus our detailed attention on the client software, since that is the primary contribution of this paper.

4.1 Generic model

First, we describe a generic model for a streaming media system. The basic components of such a model would include the following.

- Streaming web server
- Authentication protocol on web server and client
- Client web-browser to request media
- Client application to receive media data (e.g., Real Player)
- Client library interface between kernel (device driver) and user space (application)
- Client device driver to utilize media data

The basic components of this generic model are shown in a simplified block diagram form in Figure 1.

This generic system would function in the following manner.

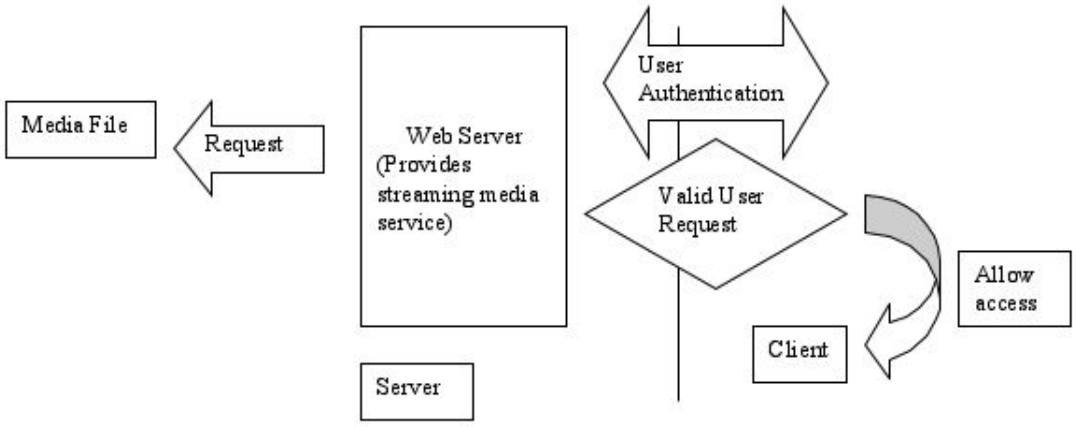

Figure 1: Generic Streaming Media System

1. The web server offers streaming media services
2. A client requests a media file from the web server
3. The web server authenticates the user and the user authenticates the web server (mutual authentication)
4. On successful mutual authentication, the web server employs RTP to stream the data from the server to the client
5. The client web browser opens the default media application (e.g., Real Player or Windows Media Player)
6. The media application strips the RTP header and sequences the packets (assuming the transport layer protocol is UDP).
7. The media application uses system calls or library functions to write the data to the device that plays the file
8. The device driver stores the data in its internal memory, using interrupts (or other procedures) to write the data to the appropriate port

The above system has several security vulnerabilities. For example, the streamed data can be captured at any point between the two endpoints and the resulting data is subject to replay. Moreover, there is no protection to prevent the client side from capturing and redistributing the data to others.

5 Proposed secure model

Our proposed security model includes the same basic components as the generic streaming media system discussed in the previous section, with a few additional security features. For example, the web server includes a License Manager to manage access to requested data. The operation of this feature will be described in more detail below.

Another security feature involves a "scrambling" algorithm, which is employed by the server, and a corresponding de-scrambling algorithm that is employed by the client. A scrambling algorithm should be unknown to a potential attacker and an attacker must be required to "break" the scrambling algorithm in order to recover an of the data. In addition, the server must have access to a significant number of distinct scrambling algorithms.

Scrambling serves two purposes. First, the scrambling algorithm creates a layer of obfuscation, making reverse engineering of the client software more difficult. Second, scrambling provides for a high degree of individualization (or uniqueness) of the client software. Consequently, scrambling algorithms that are unknown to a potential attacker are preferred.

Perhaps the ideal scrambling algorithm is a cryptosystem, since it could be applied to all of the data. However, no cryptographic algorithm is considered secure until it has undergone extensive peer review and withstood the test of time. But we will not depend on the scrambling algorithm for cryptographic strength, since we also employ standard strong encryption. Therefore, a homemade cryptographic algorithms that provides even minimal cryptographic strength will serve well as a scrambling algorithm. For example, the "tiny encryption algorithm" (TEA) [14] can be modified in many different ways to yield a large class of scrambling algorithms. While none of these modifications could be claimed to provide great cryptographic strength, each should serve well as scrambling algorithms. The rationale behind scrambling is further discussed—within the context of DRM—in [8].

Given such a set of scrambling algorithms, each client will be equipped with a subset of the available scrambling algorithms. The list of scrambling algorithms known to the client will be encrypted with a key known only to the server, and stored on the client. After authenticating the server, this encrypted list will be passed from the client to the server. When the server receives the list, the server decrypts it and randomly choose from among the client's scrambling algorithms. The ID number of the selected scrambling algorithm is then passed from the server to the client. Note that this process eliminates the need for a database containing the mappings between clients and scrambling algorithms.

By having different scrambling algorithms embedded within different clients, and by selecting at random from a client's algorithms, each client is unique, and each each communication between client and server depends not only on different keys, but also on different algorithms embedded in the client software. An attacker who is able to break one particular piece of content, will likely still have a challenging task when trying to break another piece of content destined for the same client. And even if an attacker completely reverse engineers one client, it is likely that he will still need to expend roughly the same effort to attack any other client.

On the server side, the data is scrambled, then encrypted. On the client side, the data is decrypted, and the resulting scrambled data is passed to the media application. The media application passes the scrambled data to our "secure device driver" (discussed in more detail

below), which de-scrambles the data. In this way, the data is obfuscated until the last possible point in the process.

Given these security features, the secure streaming media process proceeds as follows.

1. The secure web server offers streaming media services.
2. A client requests a media file from the secure web server.
3. The secure web server authenticates the user and the user authenticates the web server.
4. Upon successful mutual authentication, the web-server gives the IP address of the client machine and client's username to its License Manager. The client sends its encrypted list of supported scrambling algorithms to the server.
5. The License Manager verifies that the user on that particular machine is allowed access to the requested media file.
6. If the user is allowed access, the License Manager generates two random keys. The first key will be for secure RTP packet encryption (using AES) and the second key will be the scrambling key used on message blocks of media data.
7. The server generates a random number to select from among the scrambling algorithms supported by the client. It generates another random number to be used as the key for the scrambling algorithm. Both of these are encrypted (but not scrambled) and passed to the client. The client must acknowledge receipt of this information.
8. The server use cipher block chaining (CBC) to scramble the data per packet, with a randomly-selected initialization vector (IV) included with each packet (for cryptographic terminology and information, see [15]).
9. The secure RTP algorithm with the Advanced Encryption Algorithm (AES) with 128-bit key is applied to the scrambled data in each packet. The packets are CBC encrypted with a random IV included in each.
10. The scrambled and encrypted secure RTP packets are transmitted over the network.
11. The client web-browser opens the secure media application for the file.
12. The media application requests the secure RTP decryption key. The user must authenticate in order for the client to obtain the decryption key. For example, authentication could be smart-card based. Of course, any other user authentication method could be applied on the client.
13. The media application strips the secure RTP header and sequences the packets (assuming the underlying transport protocol is UDP).

14. The media application initializes its secure device driver with the ID number that specifies the scrambling algorithm used by the server, and the algorithm is initialized with the scrambling key.

15. The media application is oblivious to the scrambling. It therefore writes the scrambled data to the device that plays the file.

16. The secure device driver de-scrambles the data and writes plaintext data to the appropriate device buffer and port.

Our secure streaming media system is summarized in Figure 2.

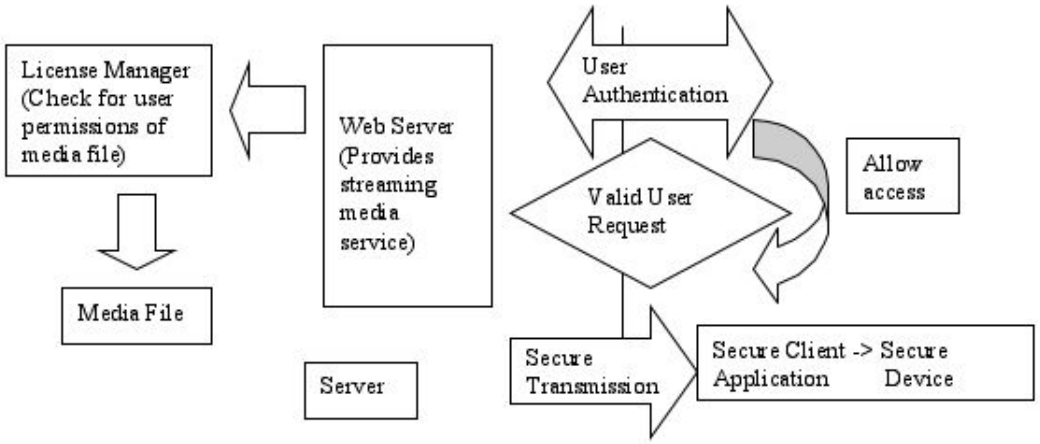

Figure 2: Secure Streaming Media System

Figure 3 gives a simplified view of the interaction between the client and server in our secure model.

6 Comparison with a typical DRM system

Here we give a brief comparison of the security features in our secure streaming media system—as presented in the previous section—with the features available in Windows Media Player [12], a typical DRM system. The following six points are listed on Microsoft's website [12] as the primary security features of Windows Media Player. We describe how our system implements each of these features.

- **Persistent Protection**: The proposed model gives an individual license key to a client on a per transaction basis for each requested file. The protection is not only over the insecure network between client and server, but, due to the scrambling and the secure device driver, it persists all the way to the clients media device.

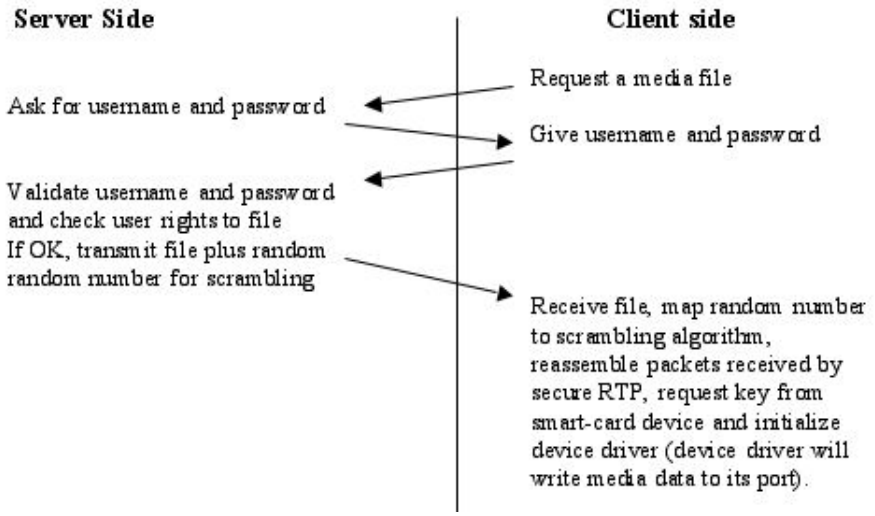

Figure 3: Proposed Secure System

- **Strong Encryption**: The proposed model uses RTP with 128-bit AES encryption. Our model also employs scrambling, but it does not rely on scrambling for cryptographic security.

- **Individualization**: The scrambling algorithm is selected at random and the actual set of available scrambling algorithms is individual to each client. Therefore, the compromise of one client does not break the entire system. Moreover, the broken client can easily be replaced with an upgraded device driver that employs a different set of scrambling algorithms. The secure device driver could be made unique in other ways as well. For example, the methods discussed in [16] or methods similar to those employed by metamorphic virus writers [17] could be implemented. Such protections would certainly make the reverse engineering problem even more challenging for an attacker. We have not implemented this higher level of uniqueness, but it would clearly be feasible to do so.

- **Secure Media Path**: Our secure model does not de-scramble the media data until the last possible point in the process. The data passes through the entire system in scrambled form. Of course, over the insecure channel it is further protected by strong encryption.

- **Revocation and Renewability**: Compromised players can be revoked by maintaining a revocation list with the License Manager. Revoked clients will then fail to authenticate with the License Manager. Moreover, if a particular scrambling algorithms is compromised, the server could simply avoid using the compromised algorithm. Alternatively, all clients using the compromised algorithm could be upgraded with new device drivers that do not include that particular algorithm.

- **Secure End-to-End Streaming and Downloads**: Secure RTP is used for end-to-end streaming and downloads. The AES encryption algorithm (or other strong encryption algorithm) is used in secure RTP. Secure end-to-end transmission can also be accomplished using other well-established methods such as IPsec [5].

As can be seen from the proposed model and the discussion above, the License Manager is a significant part of our secure streaming media system. But it should be clear that license management is not the heart and soul of the system. The secure device driver and the uniqueness achieved via scrambling are the crucial security aspects of the system.

7 Implementation issues

The operating systems commonly used in embedded multimedia devices are Real Time-Linux, Linux and VxWorks. Here we briefly compare implementation issues for the proposed secure streaming media model on these three operating systems.

VxWorks does not provide any memory protection between application and system tasks. This implies that the device driver memory buffer is available to any module through simple function calls. In our secure model, de-scrambling in the device driver adds overhead, making the process not as lightweight as in the insecure case. If the decrypted media data is made available in the memory buffer only for periodic time slices, it would make it more difficult for a hacker's processes to obtain the decrypted data in the same time slice. The hacker's process would need to synchronize with the availability of the decrypted data. This attack would be more difficult in a single processor system.

RT-Linux requires the user to divide the application into two distinct parts—the real-time part and the non-real-time part. The real-time part is serviced rapidly, allowing it to meet deadlines, while the non-real-time part has the full range of Linux resources available for use, but it cannot have any real-time requirements. The division of multimedia data into real-time and non-real-time part must be done carefully. The constraints increase if the streamed data is live and interactive. De-scrambling of media data on the device driver level increases the timing constraints on such data.

The implementation issues with the Linux operating system are simplified, since Linux provides memory protection between kernel and user space. Linux also has a non-swappable memory area for key material protection. Our prototype is Linux based.

8 Performance

We performed timing tests on the secure transmission, including our the secure device driver. The timing depends on the scrambling algorithms. The tests were performed on 1 GHz Pentium-IV PC by looping over calls to the process using C library function `clock` to make timing measurements.

The open source secure-RTP library [10] has built-in function calls to test transmission throughput for AES-128 bits encryption using TMMHv2 authentication. We included that

library in our test to calculate the transmission throughput using our secure device driver. The transmission throughput is affected by the transmission time plus the time it takes to scramble and descramble data and pass through the secure device driver.

Excluding the scrambling and secure device driver, we found that the transmission throughput is about 125 Mbits/second. This result is shown by the light colored line in Figure 4. The output saturates with increasing size of packets. If we include the secure device driver and transmission time, then the throughput we obtained is shown in the dark line in Figure 4. As expected, the proposed secure streaming media model incurs a slight performance penalty.

The cryptographic algorithms used by the server and client are the bottlenecks. But it can be seen that a reasonable data rate is available for streaming multimedia applications. If higher data rate is necessary, then cryptographic hardware accelerators could be employed to encrypt and decrypt data.

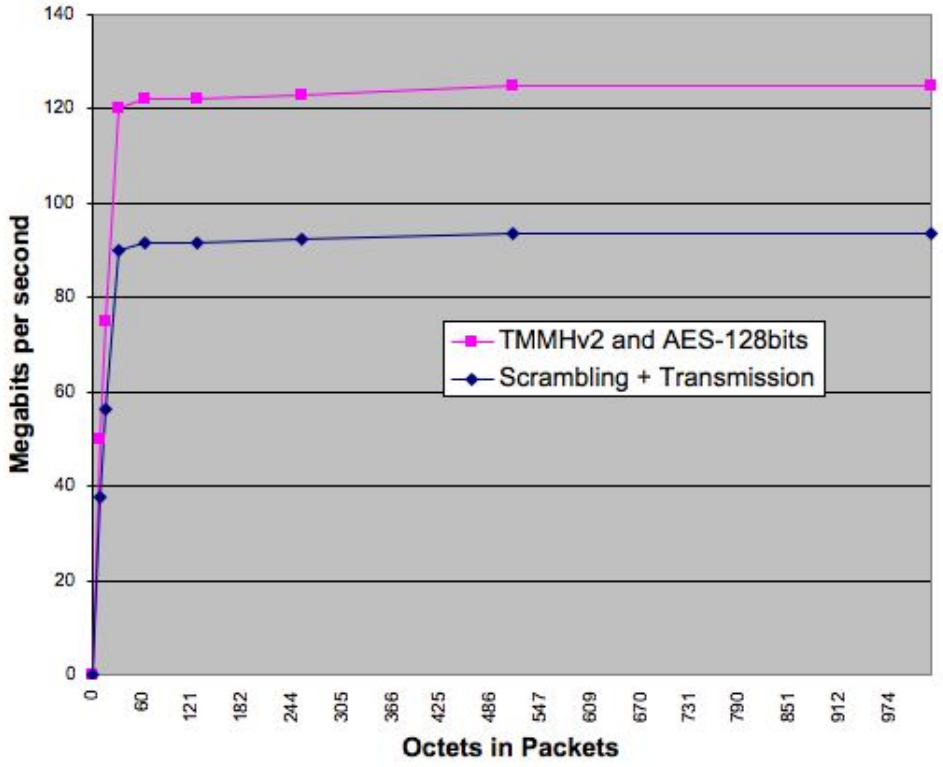

Figure 4: Timing and Performance

9 Conclusion

In this paper we presented a method for increasing the security of streaming media. Our appoach, which is based on concepts from digital rights management, adds a measure of

integrity protection, but is primarily intended to aid in replay preventions.

With any conceivable PC-based security model, a dedicated hacker can, with sufficient effort, successfully attack a particular piece of content. This is unavoidable if the media is to rendered on a system with an open architecture (such as a PC) and that system is controlled by the attacker. Under our proposed secure streaming media system, we believe the amount of work required for such an attack would be significant. However, the real strength of our approach is that the overall system will survive, even when individual pieces of content are successfully hacked.

References

[1] EFF Consensus at Lawyerpoint, Hollywood want to plug the 'analog hole', May 23, 2002, `http://bpdg.blogs.eff.org/archives/000113.html`

[2] William Stallings, *Cryptography and Network Security: Principles and Practices*, Prentice Hall, 2003.

[3] The OpenH323 Project, `http://www.openh323.org/`

[4] Media-S, Sidespace Solutions, Inc., `http://www.sidespace.com/products/medias/`

[5] Charlie Kaufman, Radia Perlman, and Mike Speciner, *Network Security: PRIVATE Communications in a PUBLIC World*, Prentice Hall, 2002.

[6] Real-time Transport Protocol, RFC 1889, `http://www.faqs.org/rfcs/rfc1889.html`

[7] The Secure Real Time Transport Protocol, IETF draft, `http://www.globecom.net/ietf/draft/draft-ietf-avt-srtp-00.html`

[8] Mark Stamp, Digital rights management: the technology behind the hype, *Journal of Electronic Commerce Research*, Vol. 4, No. 3, 2003,

`http://www.csulb.edu/web/journals/jecr/issues/20033/paper3.pdf`

[9] Ross Anderson, *Security Engineering: A Guide to Build Dependable Distributed Systems*, John Wiley & Sons, 2001.

[10] Secure RTP, `http://www.vovida.org/protocols/downloads/srtp/`

[11] Jon Crowcroft, Mark Handley and Ian Wakeman, *Internetworking Multimedia*, Morgan Kaufmann, 1999.

[12] Windows Media Digital Rights Management Offering, `http://www.microsoft.com/windows/windowsmedia/wm7/drm/offering.aspx`

[13] NIST special publications, Secret key distribution,
http://csrc.nist.gov/publications/nistpubs/800-7/node209.html

[14] TEA, a tiny encryption algorithm,
http://www.ftp.cl.cam.ac.uk/ftp/papers/djw-rmn/djw-rmn-tea.html

[15] Bruce Schneier, *Applied Cryptography*, second edition, John Wiley & Sons, 1996.

[16] Puneet Mishra and Mark Stamp, Software uniqueness: how and why, *Proceedings of ICCSA 2003*, P. P. Dey, M. N. Amin and T. M. Gatton, editors, July 2003, http://home.earthlink.net/~mstamp1/papers/iccsaPuneet.doc

[17] Ivan Balepin, Superworms and crpytovirology: a deadly combination,
http://wwwcsif.cs.ucdavis.edu/~balepin/new_pubs/worms-cryptovirology.pdf