Hello, Are You Human?

Jun Xu, Richard Lipton, Irfan Essa

College of Computing

Georgia Institute of Technology

Atlanta, GA 30332-0280

{jx,rjl,irfan}@cc.gatech.edu

Technical Report (GIT-CC-00028)

November 13, 2000

Abstract

In this paper, we propose the concept of a humanizer and explore its applications in network security and E-commerce. A humanizer is a novel authentication scheme that asks the question "are you human?" (instead of "who are you?"), and upon the correct answer to this question, can prove a principal to be a human being instead of a computer program. We demonstrate that the humanizer helps solve problems in network security and E-commerce that existing security measures can not address properly. A key component of this "are you human?" authentication process is a new type of trapdoor one-way hash function, called Turing-resistant hashing. It transforms a character string (the preimage) into a graphical form (the image) in such a way that a human being won't have any problem recovering the preimage through the trapdoor of human pattern recognition skills, while a computer program, essentially a Turing machine, will not be able to decode it or make a correct guess of the preimage with non-negligible probability. Based on this hash function, we design a stateless generic humanizer that can be parameterized for use in various real-world applications.

1 Introduction

One of the key problems in computer security is to prove "who" one is, known as authentication problem, and there is a vast literature on authentication protocols. The problem we pose and solve in this paper, however, is not to prove exactly who one is, but rather to prove that one is human, not just another computer program.

In [AN96], Abadi and Needham propose a set of principles for engineering cryptographic protocols. Their first principle is "every message should say what it means." To follow this principle, however, is not a trivial task because the semantics of a security application may not be explicit. We discovered a new semantics, "are you human?", that should be "meant" in many real-world security and E-commerce protocols, yet existing cryptographic measures do not lend us a way to "say" it. Therefore, security needs of these protocols generally are not well addressed by existing cryptographic measures.

We not only discovered a new semantics in computer security, but also constructed a novel authentication scheme, called a humanizer, to "say" it. The idea of humanizer is inspired by Turing's test for artificial intelligence. We observe that the "are you human?" problem is essentially a variation of the famous "Turing Test" of Alan Turing [Tur50]. The Turing test asks whether it is possible for a computer to fool a human into thinking that it is a human. We show that our humanizer is a very simple, yet powerful method that can easily distinguish humans from computer programs in a single test.

The key component of our humanizer is a new type of trapdoor one-way hash function called Turing-resistant hashing[1]. The hash function transforms a character string into its graphical representation (the image) in such a way that a human being won't have difficulty recovering the original character string (the preimage) while a computer program, essentially a memory-constrained Turing machine, will not be able to decode the image with high confidence within a reasonable amount of time. In the remainder of this paper, we will use the term computer program and Turing ma-

[1] Public key cryptosystem [DH76] is a well-known type of trapdoor one-way hash function.

chine interchangeably. The trapdoor to this one-way hash function is obviously the human pattern recognition skills. We will show that recovering the preimage without going through this trapdoor is equivalent to solving the Turing test in the domain of pattern recognition, which we assume to be a hard problem, if not impossible at all.

Based on this hash function, the basic idea of a humanizer protocol is quite simple. For a server A to verify that a client B is a human being rather than a Turing machine, A transforms a random string S using the Turing-resistant hash function f and transmits $f(S)$ to B. If B correctly recovers the preimage S from $f(S)$, A can be sure that B is a human being based on the assumption that B has to go through the trapdoor of human pattern recognition skills to recover the preimage. Nevertheless, there are nontrivial security and system issues that need to be addressed to make a humanizer really work and these issues will be discussed in Section 4.

1.1 Organization of the paper

The rest of the paper is organized as follows. Section 2 discusses various applications of a humanizer in network security and E-commerce. Section 3 presents our Turing-resistant hash function. Section 4 presents a generic humanizer protocol that can be parameterized for use in applications discussed in Section 2. Section 5 surveys the work related to humanizer and its applications. Section 6 summarizes this paper, reports our work-in-progress, and discusses directions for future research.

2 Applications

We discovered that "are you human?" is an important semantics in many computer security and E-commerce applications that involve the interaction between a human being and a computer server. These applications and the importance of the semantics in these applications will be discussed in detail in Section 2.1, 2.2, and 2.3.

2.1 Stop Unwelcome Screenscraping

The original meaning of a screenscraper is a computer program that parses and interprets the dumb terminal output from a mainframe transaction server and renders the result in a PC graphical user interface. Nowadays, screenscraper has been given a new meaning: a computer program that parses and interprets the data from a web server that belongs to *someone else*, and renders and/or processes it in a way the screenscraper owner desires. The latter definition will be used in the rest of this paper.

Here is an example how an unwelcome screenscraper works. Suppose Alice is the owner of a large E-commerce web site that sells hundreds of thousands of different products (e.g., books) to online customers. The server's CGI (common gateway interface) forms are designed to allow everyone in the world, a potential customer, to search for product pricing and availability from Alice's product database. Unfortunately, a screenscraper program owned by Bob, one of Alice's competitors, can access Alice's service in the following undesirable way: It sends CGI queries to Alice's server looking up price information of every single item in her big database, extracts the price from the query result, and makes his price on the same item slightly cheaper than hers, all done *automatically*. Alice certainly does not want to tolerate this kind of online price war: it consumes the CPU cycles of her web server and generates negative revenue for her!

Alice's web server is susceptible to such automatic information stealing because the protocol used in the application does not "say" exactly what it "means," a violation of the aforementioned first principle [AN96]. Alice's web server "means" to show potential *human* customers price and availability. However, rendering it in ASCII format and predictable structure actually "says" that both Turing machine (here the screenscraper) and human being are equally welcome!

It is hard to use traditional access control mechanisms to guard against such accesses. Limiting pricing and availability search capability only to registered customers is both undesirable and futile: it may scare potential customers away to have to spend time registering before they see any useful information; Bob can register as a legitimate user and run screenscraper from that account. Adhoc countermeasures such as limiting the number of accesses won't work well either since customer needs are very different and more than one account can be registered by a single person under many false names.

Since Alice's intended customers are human beings and screenscrapers are Turing machines, this problem boils down to a key question: how can Alice deny service to a Turing machine and at the same time grant service to a human being? This problem can be solved by our humanizer, which works as follows. When a customer receives a CGI form, a random short string will be supplied to him in the aforementioned hashed form. When he fills out the form he needs to recognize the string and enter in a specified field. The query will be honored only when the string submitted with the form is the correct one. This costs a human being only a few seconds to type in the string, while it denies a Turing machine even a single access!

Bob can certainly perform the screenscraping manually, but the speed would be limited to how fast he can recognize and enter these strings, which would be several orders of magnitude slower than if this is done automatically by a computer program. With a database of hundreds of thousands of products, this is going to be a tedious and backbreaking job, which few people would be willing to do it.

2.2 Prevent Brute-Forcing Passwords

Another case where it is important to prove that one is a human is in the area of login using password or PIN (Personal Identification Number). While previous work concentrates on the vulnerability of password- or PIN-based authentication protocols over an insecure channel, the problem we are interested in here is to prevent the password or PIN number from simple brute-forcing. We illustrate the problem in the context of online banking while the problem itself exists in many other password- and PIN-based protocols as well. A computer program can attempt to log in an account by trying different PIN numbers (typically 4 to 6 digits as they *have to* be easy to remember [MOV96]) thousands of times automatically. Without countermeasures, it does not take long to compromise a PIN. The good old way to counter this on ATM machines is to confiscate or lock the card after a few failed trials [MOV96]. The counterpart of this confiscation in online banking is to freeze the account. However, this leads to denial of service. Given that bank account numbers are quite predictable (e.g., contiguous) [SVK00], it is very easy for a hacker to freeze thousands of accounts in a matter of minutes with a brute-force attack. Protection based on IP address is not quite effective in solving this problem because it can be bypassed by IP spoofing [Bel90], or attacks can be launched from a large number of compromised hosts simultaneously. It is not clear whether other measures such as introducing the delay into a transaction can solve the dilemma between broken account and denial of service.

A humanizer is a good solution for this problem without side-effects. If one has to prove himself as human being before his trial of a PIN will be considered, the number of trials that can be accomplished by a hacker is effectively limited to the number of "are you human?" questions he can answer using his eyes, brain, and fingers. A legitimate customer won't bother taking several seconds to enter an answer while the poor hacker will have to spend hours or even days answering questions before he can get close to cracking a password or a PIN. To ensure maximum security, disabling the account after certain number of failed attempts is still needed. However, if the hacker again uses this to achieve the effect of denial of service, the damage would be much smaller since it would take the hacker several orders of magnitude longer to disable an account.

A humanizer can even protect previously unknown vulnerability of a password-based protocol. Dos Santos et al. [SVK00] recently discovered the following "account sweeping attack" in online banking: by fixing a "popular" PIN (e.g., 1234) and trying different account numbers with it, a hacker can break *a large number of accounts* quickly without raising a single alarm because each account only gets one failed attempt, which is below the threshold! The damage of this security hole would be several orders of magnitude smaller should a humanizer be employed in the protocol.

A humanizer is not only applicable to online banking, but also to other PIN number or password based protocols between a human being and a computer system. For example, in credit card transactions, the expiration date is very often used as the PIN number, and a humanizer would protect the credit card number and expiration date from being brute-force attacked. In summary, a humanizer offers a generic way to strengthen PIN- or password-based security protocols because it both "means" and "says" that only *humans* are allowed to log in.

2.3 Deter Denial of Service Attack

Denial of Service (DoS) attack has become a growing concern recently. The most popular type is the TCP SYN flood attack [CER96], which takes advantage of the vulnerability in the TCP/IP socket implementation. There is a fair amount of research [S^+97] and vendor [Inc97] literature on how to counter this type of attack. Related works in denial of service in general will be discussed in Section 5.

We are interested in the DoS attack in the following E-commerce context. Knowing the syntax of a CGI query of an E-commerce website, an attacker can submit a large number of bogus and time-consuming transaction requests to the server. Though these requests will not be honored due to authentication failure, service to legitimate customers will be denied or degraded. Note that such bogus transaction may come from a hacker or a competitor. With the volumes of e-commerce transactions becoming higher and higher, this type of attack may begin to pose a real threat.

A humanizer is again a good solution to this problem because the intended customer is always human[2]. The humanizer would limit the number of bogus transactions a hacker can submit to how fast he can decode the images and enter the strings.

Now we go back to the general problem of DoS attack. One reason that DoS is hard to counter is that the hacker can replicate attack programs on hundreds of victimized machines and launch attacks from these machines simultaneously, known as distributed DoS attack. If other layers of the network can stand the heat[3], a humanizer can be especially effective against this type of DoS at the application layer because the replicated programs are Turing machines instead of human beings. The humanizer strikes the Achilles's heel of the hacker: he can duplicate his programs and

[2]Here we assume that B2B (business to business) transactions will go through a secure channel instead of the interface that is offered to online customers.

[3]Meadows points out that to counter DoS attack, every step or layer in the protocol has to be robust [Mea99].

data, but not his human pattern recognition skills.

2.4 Summary of the Applications

In this section, we have shown three types of applications of a humanizer in computer security and E-commerce areas. We show that by realizing the "are you human?" semantics that is implicit in these applications through a humanizer, the security vulnerabilities in these applications are well addressed. We expect that the number of applications where humanizer would be useful will grow since many Internet applications involve the interaction between a human user and a computer system.

3 Turing-resistant Hashing

The key component of the humanizer is a trapdoor one-way hash function, called Turing-resistant hashing, that converts a string of letters and numbers into a graphical form. The resulting image should satisfy two main requirements:

1. It must be easy for a human being to correctly recover the string from the graphical image. If it requires lots of time, no one would be very happy decoding the image.

2. It must be hard for a computer to do the same thing. The computer should not be able to make a correct guess with high probability, or it is able to do so only after a lot of computation.

Conversion of a string of letters and numbers to a graphical form is easily possible and is accomplished by rendering an image with the required string. Simple principles of typographic design can be employed to ensure ease of read by humans [TM95, Coo89] and allow for aestheticly pleasing presentation of the text.

Machine interpretation of the text presented in such a graphical form is possible by using Optical Character Recognition (OCR), which is a well-developed and at present easily available technology [UMa]. However, all efforts in this area have been aimed at digitization of documents (for example [Hai96, Sri92]). Very accurate and reliable OCR in the domain that we are interested in still a very difficult task for machines.

The main reason for this is that (a) most OCR systems are specifically tuned for the type of document that will be scanned, and (b) the computation cost of OCR is quite high. This is especially true in our domain as we propose to modify the image so that it still is easy for people to recognize, but becomes inordinately difficult for computers, especially with a limited amount of time. Consider the case of a computer attempting to decipher a string of characters embedded in a graphic image with one or more of the following characteristics:

- Characters may be random and context-free, i.e., they may not make up parts of cognizable words. Since OCR typically uses context-based error correction in its text recognition [Sri92], using random words renders this capability useless.
- Some characters may be kerned together, i.e., the spaces between the characters have been removed, and non-uniform spaces might exists between other characters.
- Some characters, randomly, may be stretched and others compressed so characters are also of a non-uniform width and height.
- Characters may be composed of different fonts, styles, and sizes.
- Some characters may be rotated to the left or right, and the characters may be wrapped on a cubic spline or presented in a non-linear fashion.
- Characters may be randomly "filled" so that no character is ever like itself. Thus, even training a program to recognize the letter "A" becomes hard.
- Characters are presented in a graphic bitmapped image, but randomly located within the image. Other areas of the graphic may contain squiggles, curly-cues, circles, arcs, and various other nonsensical writings or symbols which are not cognizable as words, and thus will not be confusing to humans, but may to a machine.

- Some noise may randomly be added to the graphic image.
- A lossy image compression like JPEG can be used to reduce the redundancy (or entropy) of the image to make the OCR decision process more susceptible to error [Sri92] while making no difference to human decision process.

As a convention in cryptography, we assume that the algorithm used to morph characters into a graphic representation is public. However, this won't make the job of decoding an image by an OCR program easier due to following two reasons: (1) Most aforementioned variations can be parameterized and the composition (the product space) of these parameters can result in a huge "parameter space" to search if the hacker is determined to tune OCR for recognizing the string. (2) Any or all aforementioned variations may be randomly selected on a per character basis, and no single character may easily reappear as an exact representation of itself. Reason (1) tells us that each character will take a long time to recognize due to the huge parameter space to "guess" and the probability of guessing it right is low, say p. Reason (2) tell us that each character is independent and the probability of guessing the string right is p^N if there are N letters in the string. Also, the variations such as string in a bitmapped image background will confuse the hacker OCR program as to where the string is.

Turing-resistant hashing is a new type of trapdoor one-way hash function, in addition to its well-known cousin: public-key cryptography [DH76]. All the aforementioned random variations can be viewed as the convolutions, perturbations, and random padding engaged in cryptographic hash functions to make them one-way and collision resistant [MOV96]. This leaves one and only one trapdoor to recover the preimage: the human pattern recognition skills. The fact that a computer program, essentially a Turing machine, won't possess such level of human intelligence is backed up by a the hardness of passing the Turing test in general, and the arguments above in particular.

It is known to be simple and fast for a computer to transform a sequence of letters or numbers into a series of "randomly formed characters" according to the aforementioned random variations, and we are currently developing such a system. It is also known that existing OCR algorithms do not handle character strings with such random perturbations very well. However, it is an open question whether people can design a highly intelligent OCR program that can do a better job. In section 6, we will discuss the future work that needs to be done on this problem.

Note that our Turing test need not be perfectly correct in order for the idea to work in practice. All that is required is (1) that human beings are rarely misclassified as Turing machines and (2) the Turing machines are detected with reasonably high probability. If our implementation satisfies the first condition, then the system will be acceptable to authorized users. If our implementation satisfies the second condition, then the security threat posed by automatic attacks will be significantly reduced.

4 Cryptographic Humanizer

If we view the Turing-resistant hashing as a way to generate random puzzles only human beings can solve using their pattern recognition skills, a humanizer is a protocol that gives the puzzles to clients, checks if the puzzles are correctly answered, and most importantly makes sure no client can cheat in the process. There are two key technical issues to be addressed: (1) how does server check whether the client's answer is correct, and (2) how replay of a previous challenge/answer pair is prevented?

A simple and obvious answer to (1) is to remember answers to all the puzzles that are given out. However, this stateful approach would require the storage of state information for every outstanding puzzle including the serial number that identifies a puzzle and the answer to the puzzle. This would result in a large amount of state information to be kept on a busy server. A better alternative is a stateless (for the server) approach in which whether the puzzle is correctly solved or not can be directly inferred from the client's answer and the encrypted state information kept by the client. There are situations where the stateless approach would be ideal. For example, in E-commerce applications, one server may be responsible for generating puzzles while other servers process the transactions. So the server that gives out a puzzle may not be the server that checks if the answer is correct.

The way to make the server stateless is as follows. When a puzzle is given out, the correct answer (the string) to the puzzle, along with the information that uniquely identifies the puzzle such as serial number and timestamp will be hashed into a Message Authentication Code (MAC) using a secret key (shared among all transaction servers in the aforementioned multi-server case). We use MAC instead of digital signature [MOV96] since the latter is order of magnitude slower. This MAC will be a part of the puzzle to be given to the client and should be returned to the server along with the transaction request. In this way, the client keeps the state (the MAC) so that the server does not need to keep them. The client can not tamper with the MAC because it is encrypted using the secret key of the server. To check if the puzzle is correctly solved, the server only needs to check if the message authentication code computed from the client's answer matches the MAC received.

Now we are ready to describe our generic humanizer protocol. Here we assume that a single server processes all the transactions. As mentioned above, the stateless approach makes the multi-server design easier. Each puzzle $h(STR)$ is generated from a random string STR using the Turing-resistant hash function h. The puzzle is uniquely identified by a serial number SRN, which increases by one every time a new puzzle is given out (no wraparound). The server uses a secret key k and a keyed cryptographic hash function F such as HMAC-MD5 [BCK96, KBC97] to generate the MAC. A timestamp TMP is associated with every puzzle. The maximum lifespan of a puzzle is MLS. An answer to a puzzle that is older than MLS will no longer be honored. The current time is denoted by T. The protocol between a client and a server is executed as follows:

1. The client sends an HTTP request to the server for the CGI form DOC.
2. The server generates a random string STR, computes $h(STR)$ and $MAC = F_k(SRN \| TMP \| STR)$ and sends DOC, SRN, TMP, MAC, and $h(STR)$ to the client.
3. The client recovers STR' (may not equal to STR if the client is not human) from $h(STR)$ and sends DOC' (with fields filled out), SRN, TMP, MAC, and STR' back to the server.

4. The server rejects the query and notifies the client if any of the following happens: (1) $T - TMP > MLS$, which indicates that the puzzle has expired, (2) $F_k(SRN \| TMP \| STR')$ does not match MAC, which indicates that the puzzle has not been correctly answered, and (3) the answer is a replay, the detail of which will be discussed next. Otherwise, the server executes the query and sends the result to the client.

Now we need to address the problem of the replay of a solved puzzle. To counter replay, the server needs to remember which puzzle has been solved. The server maintains a bit array $A[0..N-1]$ of N entries. N should be made comfortably larger than maximum number of puzzles that are given out during any period of length MLS (the maximum life span of a puzzle). In protocol step two, when a puzzle with serial number $SRN1$ is given out, $A[SRN1 mod N]$ will be set to 0, indicating that it has yet to be answered. In protocol step four, when a puzzle with serial number $SRN2$ is received, the answer will be rejected if $A[SRN2 mod N]$ is 1, indicating that it has been answered. Otherwise, if the answer is honored, $A[SRN2 mod N]$ will be set to 1. It is not hard to see that the way N is chosen prevents the "wraparound" ambiguity from happening.

We recommend HMAC-MD5-32 (output truncated to 32 bits) [BCK96, KBC97] for generating the message authentication code because its (HMAC-MD5) properties are carefully validated and it can be computed very fast. However, other keyed cryptographic hash functions [PO95] with similar properties may also be used. It is reported in [Dai00] that using Dai's Crypto++ 4.0 library, a 850 MHz Intel Celeron processor can generate HMAC-MD5 for about one hundred million bytes of preimage per second. This speed is translated into generating MACs for millions of puzzles per second, which is fast enough for even a high-end transaction server. We truncate the output to 32-bit instead of the 80-bit recommended minimum in [KBC97] because (1) the birth day attack, which compromises nonrepudiation, is not an issue in this protocol, (2) 32 bit still makes the probability of "guess it right" negligible, and (3) this makes the MAC key harder to guess, as shown in [PO95].

5 Related Works

To the best of our knowledge, "are you human?" is a new semantics. The humanizer and Turing-resistant hashing that realize the semantics are also novel ideas, and we could find no directly related work in the security literature. The connection of this work to the famous "Turing test" in artificial intelligence was explained in Section 1.

There is a vast literature [Lam81, LGSN89, Bel92, GLNS93, HK98] on how to protect a password or PIN number[4], which is communicated over an insecure channel, from being eavesdropped, guessed, inferred, or in other ways compromised. We, on other hand, investigate the problem of compromising a password just by blind guessing. Our work and Dos Santos' [SVK00] show that it is possible to compromise passwords by brute-force trying and we show that the existing countermeasures will lead to denial of service.

There is a large body of literature on denial of service attack. Early works by Gligor [Gli83], Yu and Gligor [YG90], and Millen [Mil92, Mil95] treat it as a theoretical resource allocation problem in operating systems. A more recent study of it comes from Needham [Nee94], which offers a concrete example. Dwork [DN92] considers junk mail as DoS attack and proposes an effective countermeasure. The 1988 Internet worm [Spa89] is arguably the earliest DoS attack, which happened before the Internet became popular. The nature of this problem, along with many other problems in computer security, dramatically changes in the post-web era. Denial of service instances start to be reported frequently after 1996 [How98]. The most popular type of attack is the TCP SYN flood attack [CER96] and cryptographic [IET99] and noncryptographic [S^+97, Inc97] solutions have been proposed to address it. More studies have also been done recently on the general problem of DoS attack. Meadows presents a theoretical framework for analyzing the vulnerability of a security protocol to DoS attack [Mea99]. Savage et al. proposes an IP traceback scheme [SWKA00] that can be used against DoS attack.

[4]The password or PIN is presumably weak due to human's inherent incompetence to memorize random numbers [GLNS93].

Juels and Brainard's "client puzzles" [JB99] for countering a generalization of TCP SYN flood attack bears some similarity to our approach on the denial of service problem. In their scheme, a client also has to solve a puzzle, which may take about a couple of seconds, before its connection request is honored. The difference is that they use computational puzzles that aim to consume a client's CPU resource while we use "are you human?" puzzles. The weakness of using computational puzzles lies in the fact that the speeds of computers can differ by orders of magnitude. If the slowest client (say on a 386) has to be able to solve the puzzle in a few seconds, a hacker with a high-end workstation or even a cluster will be able to solve the puzzle in a few milliseconds and will still be able to cause denial of service. From the resource allocation point of view, our approach taxes the ultimate resource of a hacker, the speed he can perform I/O as a human.

6 Summary and Future Work

We discovered that a key problem in computer security is to distinguish two classes of transactions: those generated by human beings and those generated by computer programs. We demonstrated through several real-world security and E-commerce applications that this problem is not well addressed by existing cryptographic measures, thus leading to security vulnerabilities in these applications. This motivates us to propose a a novel authentication protocol, called humanizer, that can tell a human being from a computer program in an efficient and reliable way. A key component of this "are you human?" authentication process is a new type of trapdoor one-way hash function, called Turing-resistant hashing. It transforms a string of characters (the preimage) into a graphical form (the image) in such a way that a human being won't have any problem recovering the preimage through the trapdoor of human pattern recognition skills while a computer program, essentially a Turing machine, will not be able to decrypt it or make a correct guess of the plaintext with non-negligible probability. Based on this hash function, we design a generic humanizer that can be parameterized for use in different real-world applications. Given that many security and E-commerce applications involve the interaction between a human being and a computer, we expect the humanizer to find more and more applications in the future.

We are currently designing two systems simultaneously to play the "cat and mouse" game. One is a system (the "mouse") that morphs a character string into a graphical image according to the variations described in Section 3, and the other (the "cat") is a custom OCR program that can deal with such "irregular" random string with higher precision than general OCR programs. We plan to quantify the difficulty of string recognition using OCR based on the interaction between these two systems. Our indirect contribution here is to create a positive sum game between the security community and the artificial intelligence community. The security community wins this game if they can formalize "unsolved" or poorly understood aspects of human intelligence in the area of pattern recognition. Conversely, the artificial intelligence community wins the game if they can solve the most difficult aspects of intelligence. The expected outcome of such a game is both improved artificial intelligence systems and an accurate understanding of the limits of such systems.

References

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transaction on Software Engineering*, 22(1):6–15, January 1996.

[BCK96] M. Ballare, R. Canetti, and H. Krawczyk. Keyed hash functions and message authentication. In *Proceedings of Crypto'96*, pages 1–15, 1996.

[Bel90] S. Bellovin. Security problems in the tcp/ip protocol suite. *ACM Computer Communication Review*, 19(2), 1990.

[Bel92] S. Bellovin. Password-based protocolss secure against dictionary attacks. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, 1992.

[CER96] CERT. Tcp syn flooding and ip spoofing attacks. Advisory CA-96.21, September 1996.

[Coo89] M. Cooper. Computers and design. *Design Quarterly*, 142:22–31, 1989.

[Dai00] W. Dai. *Crypto++ 4.0 Benchmarks*, June 2000.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[DN92] C. Dwork and M. Noar. Pricing via processing or combatting junk mail. In E. Brickell, editor, *Proceedings of Crypto'92*, pages 139–147. LNCS 740, 1992.

[Gli83] V. Gligor. A note on the denial-of-service problem. In *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, pages 139–149. IEEE Computer Society Press, 1983.

[GLNS93] L. Gong, M. Lomas, R. Needham, and J. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.

[Hai96] S. Haigh. Optical character recognition (ocr) as a digitization technology. *Network Notes 37*, 1996.

[HK98] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. In *5th ACM conference on computer and communication security*, pages 123–131, November 1998.

[How98] J. Howard. *An Analysis of Security Incidents on the Internet.* PhD thesis, Carnegie Mellon University, August 1998.

[IET99] IETF. *Photuris: Session-Key Management Protocol*, March 1999.

[Inc97] Checkpoint Inc. Tcp syn flooding attack and the firewall-1 syndefender. http://www.checkpoint.com/products/firewall-1/syndefender.html, 1997.

[JB99] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proc. of NDSS'99*. Internet Society, March 1999.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication.* Network Working Group, February 1997.

[Lam81] L. Lamport. Password authentication with insecure communication. *Communications of ACM*, 24:770–772, 1981.

[LGSN89] T. Lomas, L. Gong, J. Saltzer, and R. Needham. Reducing risks from poorly chosen keys. In *Proceedings of the 12th ACM Symposium on Operating System Principles (SOSP)*, pages 14–18, December 1989.

[Mea99] C. Meadows. A formal framework and evaluation method for network denial of service. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, Mordano, Italy, June 1999.

[Mil92] J. Millen. A resource allocation model for denial of service. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 137–147. IEEE Computer Society Press, 1992.

[Mil95] J. Millen. Denial of service: A perspective. *Dependable Computing for Critical Applications 4*, pages 93–108, 1995.

[MOV96] A. Menezes, P. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Nee94] R. Needham. Denial of service: An example. *Communications of ACM*, 37(11):42–46, 1994.

[PO95] B. Preneel and P. Oorschot. Building fast macs from hash functions. In *Proceedings of Crypto'95*, pages 1–14. Springer-Verlag, 1995.

[S^+97] C. Schuba et al. Analysis of a denial of service attack on tcp. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 1997.

[Spa89] E. Spafford. The internet worm incident. In *1989 European Software Engineering Conference (ESEC 89)*. Springer-Verlag, 1989.

[Sri92] S. Srihari. High-performance reading machines. *Proceedings of the IEEE*, 80(7):1120–1132, July 1992.

[SVK00] A. Dos Santos, G. Vigna, and R. Kemmerer. Security testing of the online banking service of a large international bank. In *Proceedings of the First Workshop on Security and Privacy in E-commerce*, November 2000.

[SWKA00] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for ip traceback. In *Proceedings of ACM SIGCOMM'00*, pages 295–306, Stockholm, Sweden, September 2000.

[TM95] J. Tschichold and R. McLean. *The New Typography : A Handbook for Modern Designers*. University of California Press, October 1995.

[Tur50] A. Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

[UMa] Document and image understanding image server. http://documents.cfar.umd.edu.

[YG90] C. Yu and V. Gligor. A specification and verification method for preventing denial of service. *IEEE Transaction on Software Engineering*, 16(6):581–592, June 1990.