Proceedings of The National Conference
On Undergraduate Research (NCUR) 2015
Eastern Washington University, Cheney, WA
April 16 - 18, 2015

Techniques to Enhance Security of an Authentication Protocol

Brandon Barker and Michael Smith
The Computer Science Department
Boise State University
1910 University Drive
Boise, Idaho 83725 USA

Faculty Advisors: Liljana Babinkostova Ph.D, Marion Scheepers Ph.D

Abstract

Universal Serial Bus (USB) ports in computers are commonly used to interact with a range of peripheral devices. Dongles, frequently used to control user-access to networks or software, are such devices. To ensure that only authorized computers may gain such access, a dongle must exchange data with a host computer to determine whether the computer should be authenticated. Since dongles are primarily meant to serve as a security measure, the transparency of their interactions with computers and their effectiveness in safeguarding key components of the authentication protocol are of much concern. This paper investigates the security of a dongle in the case when the authentication schema is based on linear or exponential congruences supplemented with a hashing schema based on simplified versions of the Advanced Encryption Standard (AES) or Data Encryption Standard (DES).

Keywords: Merkle-Damgård construction, authentication, dongle

1. Introduction

Internet use is now ubiquitous. Most computer users are familiar with the process of ”logging on”. The USB dongle is a common option among many for internet access. It is often used with devices such as laptops, for example. The dongle is a small hardware based security device that easily attaches to an I/O port on a computer. In addition to being devices for secure internet access, dongles are also used to prevent unauthorized access to computer resources. An in depth description of the need for such devices is given in the “Background of the Invention” section of U.S. Patent US 4168396 A .[1] An actual example of a commercial loss due to the absence of authorization checks is described in [12].

A dongle is the cyber-version of a key and uses authentication credentials and protocols embedded in the key to control access to software applications or data. Even when a user has full access to their computer, a specific software application on it protected by a dongle will not run unless the authorizing dongle is recognized by the computer. The key to the success of this schema is to make sure that the authorization to run the protected software cannot be undermined. This consideration calls for a reliable authentication schema between the dongle and the computer, as well as for encryption of sensitive information on the dongle (e.g., [19]). Security deficiencies in dongles continue to be present. For example, just recently a group of researchers from University of California, San Diego discovered [11] a serious weak point in vehicle security that allows hackers to take remote control of a car thanks to the dongles that are connected to the vehicles. In this paper another example of a security failure is exposed, and a potential remedy is given.

The process of establishing communication between a PC and a dongle requires a method for generating pseudorandom numbers. The Linear Congruential Sequence is a method for generating pseudorandom numbers (see, e.g. [14], [15], and [22]) using the linear congruence $L_{n+1} = (a \cdot L_n + c) \mod m$ and a seed value L_0. This particular method for generating the pseudo-random numbers in an authentication protocol between a dongle and host computer has been used in commercial products, as described in [12]. In [13] it was argued that to undermine this protocol would require impractical resources. However, we show that the pseudorandom nature of Linear Congruential Sequences, implemented as described in [12] and [13] without information distortion provided by hashing or encryption does not provide any security of the dongle. Then as a remedy, we consider authentication schemas based on linear or exponential congruences, supplemented with a hashing schema based on the Advanced Encryption Standard (AES) or Data Encryption Standard (DES) ciphers. The paper provides a brief description of the AES and DES encryption schemas. An extensive description of these two ciphers can be found in many papers, see for example [1], [3], [9], [23], and [25]. The paper is organized as follows: In Section 2 we give some preliminaries and motivation for this research. In Section 3 we introduce the relevant concepts regarding hash functions, give a review of a standard method of constructing a hash function from a block cipher and give a brief description of the two specific block ciphers we used in constructing hash functions. In Section 4 we describe our experimental design work. In Section 5 we report on our experimental findings. Section 6 is a discussion of the target problem in light of our findings. Section 7 concludes this study with future plans.

2. Preliminaries

Authentication protocols are often based on the principle that proof of knowledge of a privileged piece of information, or proof of ability to calculate a value requiring privileged information, identifies the party being authenticated. For such protocols to have value, it is important that the privileged piece of information not be compromised or revealed during the authentication steps. This question arose in the context of a dongle - a content protection device which, when attached to a computer, unlocks software functionality or copy-protected data. When used as a software protection device, dongles mostly appear as two-interface security tokens with transient data flow that does not interfere with other dongle functions and a pull communication that reads security data from the dongle. Without the dongle, the software may run only in a restricted mode, or not at all. The device contains a cryptographic key which is physically hidden (not visible to the end user), but which is transmitted for each authentication.

According to a 2011 study conducted by the Business Software Alliance, copy and resale of software was an estimated $63 billion dollar per year industry at that time [4]. Dongles remain a commercially viable option for copy protection of software, especially for software at a higher price point and smaller customer base [5], but there exists no standardized measure or certification of their level of security. This can lead to a flawed or false sense of security in products marketed for such purposes. It is generally assumed that values communicated between a dongle and host will be manipulated by a strong cryptographic schema such as AES [20]. However, unique schemas applied may appear to provide substantial security while further analysis reveals the provided security is relatively weak.

Consider an authentication protocol based on the linear congruence of the form $y = A \cdot x + b \mod n$ where A, b, and n are fixed, with n known publicly, but A and b, both less than n, *not* publicly known and x is generated by a system clock at the time the authentication is initiated. For example, this schema was used in a commercial product as described in [12] and [13]. In an authentication protocol relying on communicating challenge x and response y over insecure channels, an eavesdropper that can derive the values of A and b could become a successful impostor during the authentication protocol by communicating the correct response y to any challenge x. Classical methods (see e.g. Chapter 5 of [2]) allow us to efficiently derive the A-value and b-value from knowing the values of the challenge x and corresponding response y for a small number of such challenge-response rounds. The fundamental weakness of this protocol is that the it is no more difficult to undermine it than it is to solve for x in a linear congruence of the form $c \cdot x = d \mod m$: There are highly efficient methods of solving such linear congruences.

Theorem 1. *Let c, d be any integers and let m be a positive integer and $g = gcd(c, m)$. If g does not divide d then the linear congruence $c \cdot x = d \mod m$ has no solutions. If g divides d then the linear congruence $c \cdot x = d \mod m$ has exactly g nonnegative solutions less than m.*

Moreover, solutions to a solvable linear congruence can be calculated very efficiently using the classical Euclidean algorithm. Thus, an eavesdropper can successfully derive enough information from multiple executions of the authentication protocol over an insecure channel. For example, after two executions of the authentication protocol the values of the pairs (x_1, y_1) and (x_2, y_2) are known. But, then by subtracting the equations $y_1 = A \cdot x_1 + b \mod n$ and $y_2 = A \cdot x_2 + b \mod n$ and applying the theorem above (by using the Euclidean algorithm) we are able to find A and from that find the value of b within seconds[2]. This shows that the Linear Congruential Sequence does not provide any security if used without any encryption or hashing. However, even in that case we still need to know to what extent the eavesdropper can be foiled by instead communicating appropriately manipulated values of x and y. In this paper we investigate to what extent impersonation strategies can be foiled by communicating values of x and y, manipulated by certain hash functions prior to communication. Since dongles are mass produced, it is also of interest to not be able to use patterns in data collected from various dongles' authentication sessions to impersonate some of these dongles some of the time.

3. Hash Functions

Authentication protocols are often based on the principle that proof of knowledge of a privileged piece of information, or proof of ability to calculate a value requiring privileged information, identifies the party being authenticated. For such protocols to have value, it is important that the privileged piece of information not be compromised or revealed during the authentication steps. This is handled by applying what is called a "hash" function to an artifact constructed from this privileged information, and to instead communicate the "hash value". Let $\mathcal{X}$ denote the set of all possible artifacts constructed from privileged information. Elements of $\mathcal{X}$ are conceived of as strings of arbitrary finite lengths. Let $\mathcal{Y}$ be the set of hash values. The elements of $\mathcal{Y}$ are also sequences but of a fixed length, constructed from a specific finite set of symbols. A function $H : \mathcal{X} \to \mathcal{Y}$ that takes an input of variable length and produces an output of a fixed length is called a *hash function*. For a hash function to be useful in authentication protocols, it should be computationally infeasible to find for given hash value $y \in \mathcal{Y}$ a corresponding $x \in \mathcal{X}$ such that $H(x) = y$. Under such circumstances, confidence in being in possession of an x that would produce the given y, is high and the hash function has authentication value. The first informal definition of a one-way hash function was given by R. Merkle [16, 17] and M. Rabin [21].

Definition 1. *A hash function $H : \mathcal{X} \to \mathcal{Y}$ is a* **one-way hash function** *if for each $y \in \mathcal{Y}$ it is computationally infeasible to find an x such that $H(x) = y$.*

By hypothesis we have $|\mathcal{X}| > |\mathcal{Y}|$. Thus, it is guaranteed that for any function $H : \mathcal{X} \to \mathcal{Y}$ there are distinct $x,\ x' \in \mathcal{X}$ such that $H(x) = H(x')$. This fact raises concerns about the following potential weaknesses of an authentication protocol: If the protocol binds the party being authenticated to a contract, the protocol admits ability to repudiate: The party in possession of an x for which $H(x) = y$, may claim that the authentication actually binds another party instead to the contract, namely the party in possession of $x' \neq x$ for which $H(x') = y$. Thus, for high confidence in the authentication protocol based on hash function H, it must be for any $x \in \mathcal{X}$ computationally infeasible to find a $x' \in \mathcal{X}$ with $x \neq x'$, but $H(x) = H(x')$. The first formal definition of the notions below were given by I. Damgård [19, 8].

Definition 2. *A hash function $H : \mathcal{X} \to \mathcal{Y}$ is* **second pre-image resistant** *if for each $x \in \mathcal{X}$ it is computationally infeasible to find an $x' \in \mathcal{X}$ such that $H(x) = H(x')$.*

Definition 3. *A hash function $H : \mathcal{X} \to \mathcal{Y}$ is* **first pre-image resistant** *if finding an $x \neq x' \in \mathcal{X}$ such that $H(x) = H(x')$ is computationally infeasible.*

A hash function that meets these criteria is considered a *cryptographic hash function*.

3.1 The Merkle-Damgård Construction

Multiple methods for constructing hash functions have been used in practice. We feature here the method known as the Merkle-Damgård [6, 18] construction, illustrated in Figure 1. The Merkle-Damgård construction is based on a cryptographic block cipher, say F, of the user's choice, as follows:

The block cipher specifies key lengths, k, encryption block lengths e, and produces ciphertexts of length k. An initial fixed key K of length k is specified for the Hash function standard. The message M to whose hash value is to be computed, is partitioned into blocks of length e each. The final block may be too short, in which case it is padded on the left with zero-value bits so that the padded segment is of length e.

Starting with initial key K, and encrypting each message block in succession using the encrypted value produced by the prior step as key for the next, the hash value of the input message is the output, H, of the final step. It is common to refer to the block cipher F in the context of the Merkle-Damgård construction as *the compression function*.

In our application of the Merkle-Damgård schema, we shall consider for the compression function F in one instance DES, and in the other instance AES.

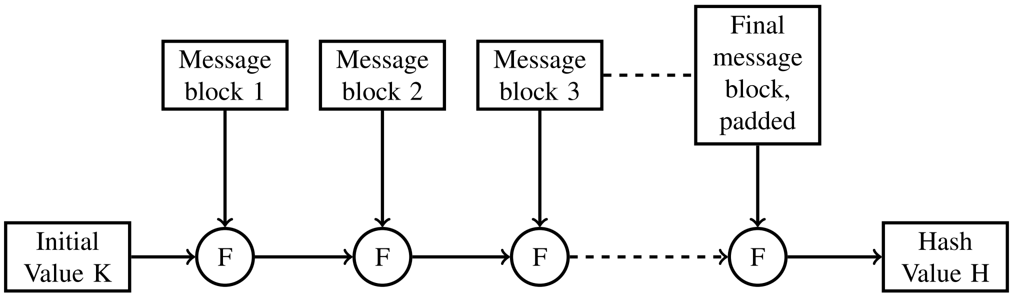

Figure 1. Merkle-Damgård Construction

3.2 Data Encryption Standard

The Data Encryption Standard (DES) is a block cipher that is structured around a Feistel Network [10]. Developed in the early 1970's, it was an official US government standard for more than 20 years. DES and its variants are still commonly used in electronic financial transactions, secure data communications, and the protection of passwords or PINs.

The Data Encryption Standard is implemented with 64 bit block sizes and 56 bit keys, keeping all values in a binary format. The full description of DES is given in [9]. For our experimental DES-based hash function we use a simplified version of DES introduced in [3], called E-DES. The novel construction of the E-DES block cipher is that it uses a different group operation (addition in $\mathbb{Z}_3$) than the usual XOR operation in DES. The E-DES uses 16 trit keys, 18 trit block sizes, and uses all values in a ternary format. Additionally, the E-DES is based upon elliptic curves as opposed to classical DES. Though not as secure as DES, the fundamental structure of E-DES is similar enough for an equivalent analysis upon its full implementation.

E-DES operates by first applying an initial permutation, referred to as IP, to the given plaintext. Next, the message is split into two halves with each half processed separately from one another. Each encryption round processes one of these halves through a Feistel function. This Feistel function is comprised of four main parts:

(1) Expansion - Expands the message half into a size that coincides with the key size.
(2) Key Mixing - A subkey generated from the original key is then added to the message using the underlying group operation in the cipher.
(3) Substitution - The message is divided into smaller blocks and processed through 3 substitution boxes.
(4) Permutation - Permutes the output to give the appearance of a random distribution from each substitution box.

E-DES performs two rounds using this Feistel function before applying a final permutation (FP). This permutation is an exact inverse of IP and thus does not contribute to the systematic scrambling performed by E-DES.

3.3 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is a block cipher based on Substitution-Permutation Network with both a variable block length and a variable key length. The versions for the block size of 128 bits and key length of 128, 192, and 256 bits were adopted by the NIST as the Advanced Encryption Standard (AES) in 2002. This cipher has a highly algebraic structure. The cipher transformations are based on operations of the finite field $\mathbb{GF}(2^8)$. Full description of the AES can be found in [1].

As we are interested only in vulnerabilities arising from exploits not requiring brute force, we considered the simplified version of AES that was introduced in [25], called mini-AES. Although the mini-AES cipher uses small-size (16-bit) keys, the underlying structure of the cryptosystem is the same as the one in AES. In our AES-based experiments, the mini-AES cipher served as a compression function in the Merkle-Damgård construction of hashing.

A round of the Mini-AES encryption applies the following functions sequentially: NibbleSub (γ), ShiftRow (π), MixColumn (θ), and KeyAddition (σ).

The **NibbleSub** function does nibble-by-nibble substitution during the forward process. This step consists of using the mini-AES S-box to find a replacement nibble for a given nibble in the input state array. The entries in the lookup table are created by using the notions of multiplicative inverses in $\mathbb{GF}(2^4)$.

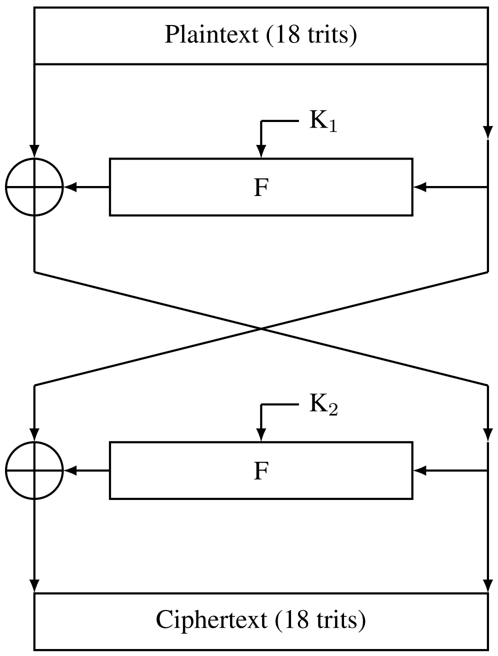

Figure 2. E-DES Structure

The **ShiftRow** function shifts the rows of the state array in the 2x2 lookup table during the forward process.

The **MixColumn** function mixes up of the nibbles in each column separately during the forward process. The mixing is created by using the notions of multiplying matrices where each matrix element comes from $\mathbb{GF}(2^4)$.

The **AddRoundKey** function adds the round key (2x2 look-up table) to the output (which is also a 2x2 look-up table) of the previous step during the forward process. The operation is similar to standard addition of polynomials (elements in $\mathbb{GF}(2^4)$) where the addition of the corresponding coefficients is performed modulo 2.

A complete mini-AES encryption is accomplished by applying two such rounds, but excluding MixColumn from the last round and including one extra KeyAddition before the first round. Figure 3 illustrates the structure on mini-AES encryption.

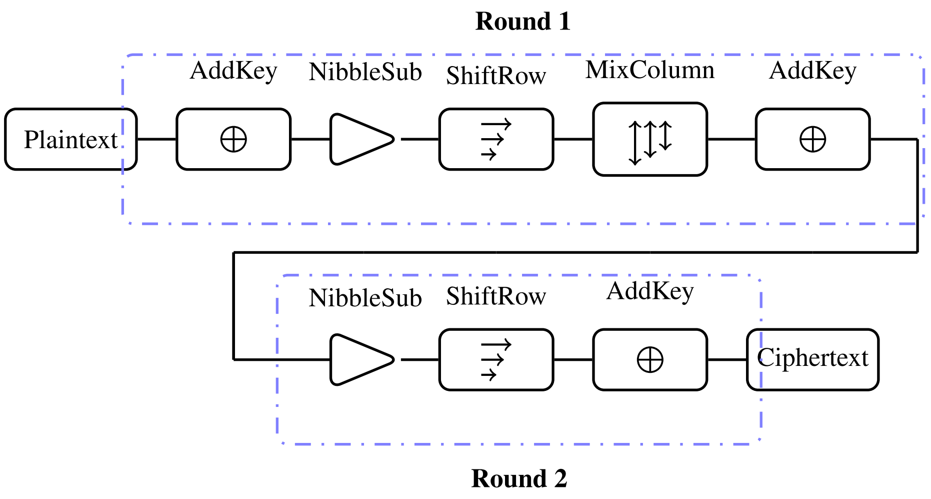

Figure 3. A schematic of mini-AES

The complete encryption can be denoted as a composition of the these functions as follows:

$$mini - AES_K(x) = (\sigma_{K_2} \circ \pi \circ \gamma \circ \sigma_{K_1} \circ \theta \circ \pi \circ \gamma \circ \sigma_{K_0})(x)$$

Here K_0, K_1, and K_2 are subkeys generated from a given 16-bit key K using a key scheduling function as described in [25]. The symbol "$\circ$" denotes the composition of functions where the order of execution is from right to left.

4. Experimental Design

We developed, in the Java programming language, software that:

- Generates data sets based on user-specified inputs of a range of x values, a value for n, and an initial A value representing a hypothetical SmartDongle$^{\text{TM}}$ of interest,
- Implements E-DES and mini-AES, and
- Implements a Merkle-Damgård hash structure based on E-DES and on mini-AES.

For data generation the software calculates from input values A and n

(1) Using a selected function[3] $f_{A,n}$, for a series of input values x the corresponding values $y = f_{A,n}(x)$;
(2) For a user specified (x, y) from the obtained series of x and corresponding y values, a user-specified number of alternative A values that would also yield the corresponding pair (x, y).
(3) For the original series of input values x, and for the newly discovered A values, corresponding y values;
(4) For all series of (x, y) values obtained $(x, Hash(y))$ where $Hash(y)$ is the result of applying in one instance, the E-DES based Merkle-Damgård hashing schema, and in the other instance, the Mini-AES based Merkle-Damgård schema.
(5) Exports all the generated data to graphing software for visualization.

Toward exploration of the influence of the modulus n, this number was chosen in some cases to be a product of two odd prime numbers, and in other cases to have several factors.

To explore the influence of the fundamental function $f_{A,n}$, it was chosen in one experiment to be the linear congruential generator, $f_{A,n}(x) = A \cdot x \mod n$, and in another experiment to be the exponential congruential generator, $f_{A,n}(x) = A^x \mod n$.

In light of findings from applying a single hashing operation, the effect of applying multiple hashing operations which varied by input parameter was explored.

5. Experimental Data

The plot of (x, y) values from the input series x, and input A, using equation $y = A \cdot x \mod n$ is predictably linear. For the alternative "A"-values obtained from the series of (x, y) computed from the input A, the (x, y) series computed from this alternative A-value collides with the original series whenever x is a multiple of the smaller factor p of n where $n = p \cdot q$ with $p<q$: This phenomenon is illustrated in Figure 4, where all the graphs collide when x is , 10, 15, 20. Note that the data is discrete: Lines are drawn to illustrate the patterns produced. Such a collision, say at $x = 5$, means that the y value at $x = 5$ is the same for all these dongles. Thus, an attacker can successfully pass the authentication test of all these dongles simultaneously when the challenge x is 5.

As illustrated in Figures 5 and 6, application of either hash function to the original y values eliminates the linear behavior of the data. However *all the collisions* identified in Figure 4 are *still* present at the same x values. These (x, y) pairs at collision sites before hashing, produce the $(x, Hash(y))$ pairs at the post-hashing collision sites. Thus, an attacker, viewing the $Hash(y)$ value for such an x in the authentication protocol of one of these dongles, may use this value to successfully pass the authentication protocol for any other dongle for which there is a collision at this x. While these graphs are drastically different in their data points, the collisions occur on the same x values.

Thus, collisions are created by the generator $f_{A,n}$, not with subsequent hashing of y. We repeated these experiments with the exponential generator, $y = f_{A,n}(x) = A^x \mod n$. Since a subsequent hashing does not eliminate collisions, we display data from this experiment for only one hash function. Results are dependent on the factorization of n.

$\underline{n = 2 \cdot 3 \cdot 24571:}$ As shown by Figure 7, unlike the case of the linear congruential generator, for the exponential congruential generator there were no collisions of (x, y) values that included every newly discovered A value. Collisions that did occur were only at even x values.

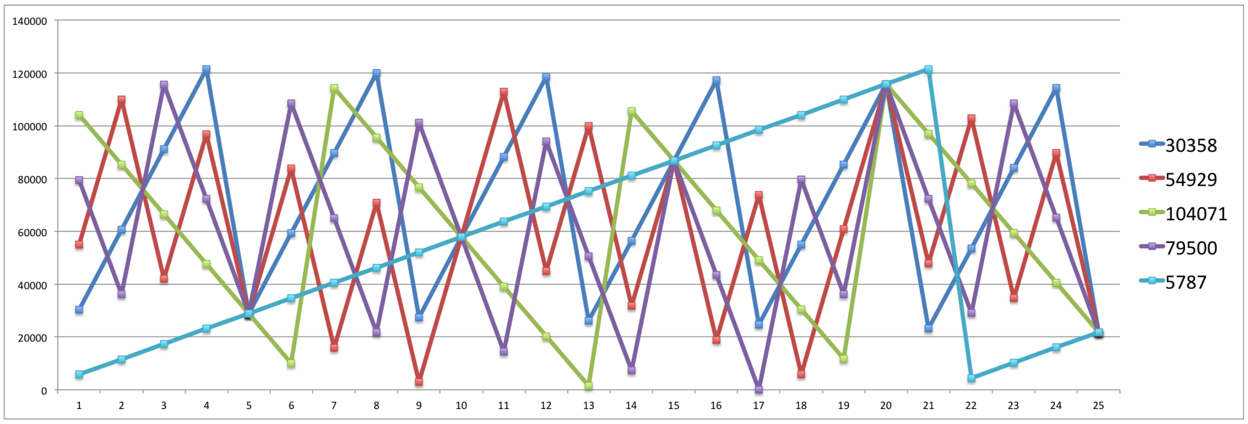

Figure 4. Linear Equation. $p = 5$ and $q = 24571$. Collisions occur at every multiple of 5. The colors indicate different A values.

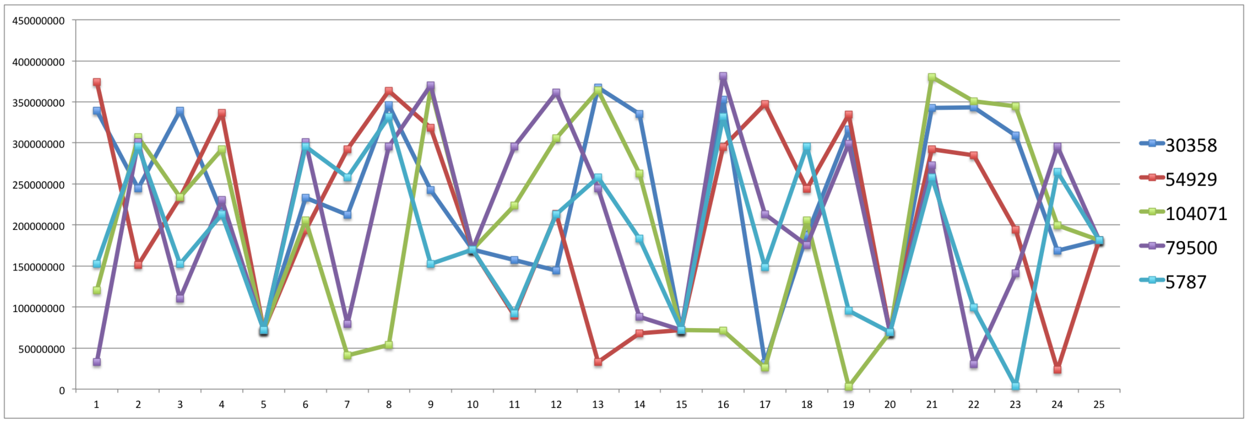

Figure 5. Linear equation transformed by an application of E-DES based Merkle-Damgård hash on y.

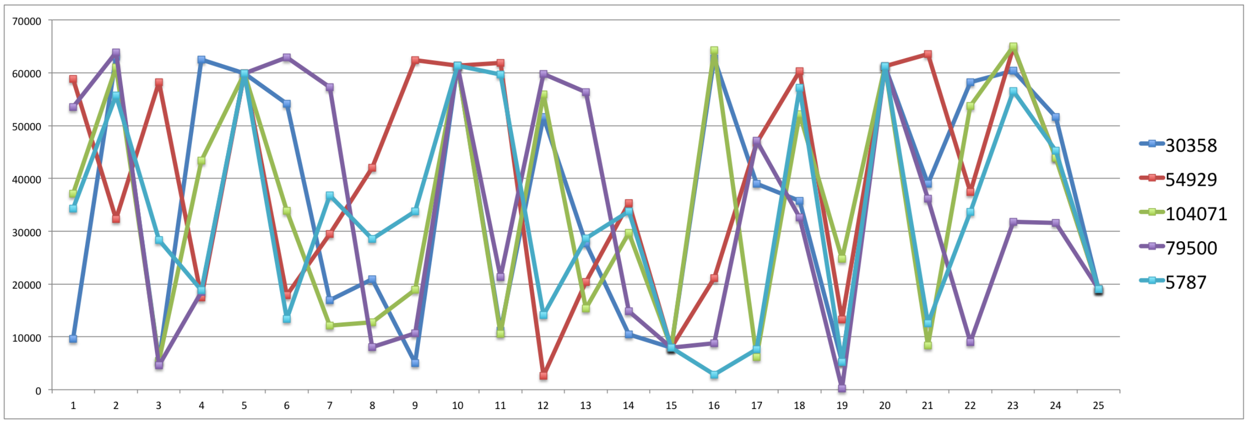

Figure 6. Linear equation transformed by an application of mini-AES based Merkle-Damgård hash on y.

$\underline{n = 2^k \cdot 24571 \text{ or } n = 3^k \cdot 24571:}$ Notably, in these cases, there were significantly more collisions observed across nearly all x values.

$\underline{n = 5 \cdot 24571:}$ As illustrated in Figure 8, nearly "total" collisions occur at all $x = k \cdot (p - 1)$ and collision pairs occur at all $x \cdot k : x|(p - 1)$ for $k>0$. (Note that $x = 12$ does not show a "total" collision. One value hashed close to the other values, but was not included in the collision).

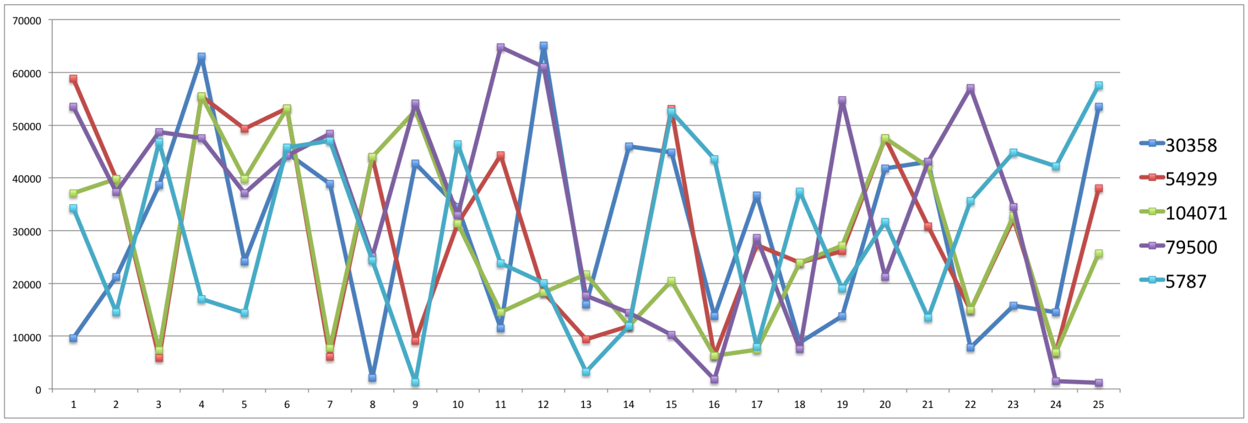

Figure 7. Exponential equation with one mini-AES based Merkle-Damgård hash. $n = 2 * 3 * 24571$.

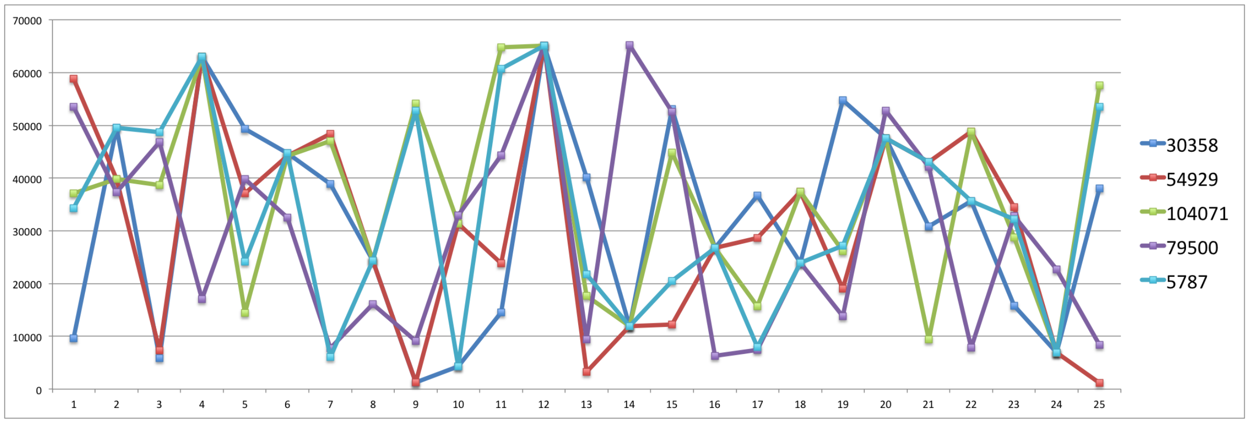

Figure 8. Exponential Equation with one mini-AES based Merkle-Damgård hash. $n = 5 * 24157$.

The observed general trend for the exponential congruential generator indicates that for non-prime squarefree modulus n at the great majority of x values the newly obtained A values produce more than one y value, while at any even x value at least one collision occurs.

These experimental results suggest moduli n at which the exponential congruential generator is an improvement over the linear congruential generator. The significant increase in computational time to generate the data for the exponential congruential generator suggests greater computational cost for the putative attacker attempting to undermine the authentication protocol. Nevertheless, the presence of collisions still represent a significant security concern as an attacker may use data collected from some devices to gain unauthorized access to other SmartDongle$^{\text{TM}}$ devices.

The third array of experiments explore hashing each output y value a number N of times, where N is based on the input parameters. For the experiments we used input parameter A, and another chosen parameter z. N is $Q+1$, where $Q = A \mod z$. Figure 9 illustrates that this multiple hashing schema dramatically alters the collision profile for the linear congruential generator experiments depicted in Figure 4 and collisions are significantly reduced.

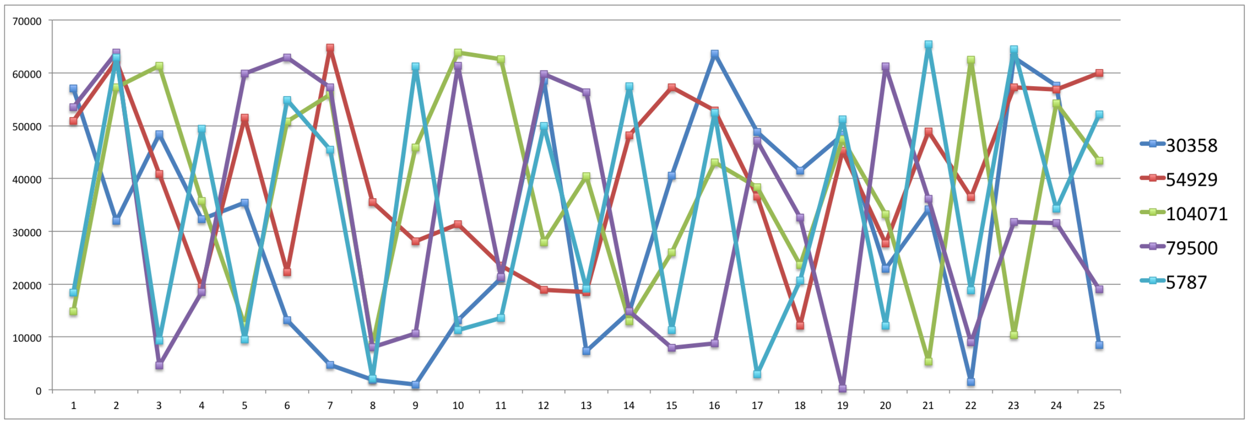

Figure 9. Linear Equation. $z = 5$. The number of mini-AES based Merkle-Damgård hashes of y is $A \mod 5 + 1$.

6. Concluding Remarks

The Linear Congruential Sequence method applies some of the strategies suggested in [20] to bolster the security of the challenge-response aspect of dongle authentication, such as ensuring unpredictability and good distribution in generated x values. However, without subsequent transformation of the produced y values, a passive attacker can discover the authenticating parameters and subsequently reliably “impersonate” the targeted dongle. Thus, the hashing accomplished by this method does not meet the criteria to be considered cryptographic hashing (one-way, second pre-image resistant, and first pre-image resistant). Applying a single cryptographic hash to the produced y curbs the ease with which an attacker could discover the authenticating parameters of a single dongle, but still leaves open significant opportunity to exploit collisions in the authenticating data stream from several devices to undermine the authentication system. Replacing the linear generator with an exponential generator impedes the computational ease with which the authenticating parameters of a single dongle could be discovered, but does not remove attacks based on collisions in the authenticating datastream from several devices. Applying a schema of variable rounds of cryptographic hashing significantly reduces the level of collisions in the authenticating data stream.

7. Future Work

This experimental exploration indicates that the base function $f_{A,n}$, characteristics of the parameters A and n, as well as a parameter-based multiple hashing all affect the reliability of the explored authentication schema. Dongles typically have a cryptosystem and cryptographic keys on board, so that the Merkle-Damgård schema could be easily deployed to augment the security of the authentication protocol. An additional feature that is likely to reduce data stream collisions among several devices is to *encrypt* the x passed from the computer to the dongle during authentication. By doing this, no value being passed from either device will be in plaintext. Multiple hashing has computational cost. Alternatives to such a schema should be explored, especially in light of the rise of small mobile devices with limited energy resources, but need of secure authentication protocols.

8. Acknowledgements

Funding for this research was provided by the National Science Foundation under award number DUE 0856815. The authors would also like to thank to Catherine Henry for her initial work upon this project.

9. References

[1] “Advanced Encryption Standard”, FIPS-Pub.197. National Bureau of Standards, U.S. Department of Commerce, Washington D.C., (2001).

[2] G.E. Andrews, *Number Theory*, Dover Publications, Inc., (1994).

[3] L. Babinkostova, A.M. Bowden, A.M. Kimball & K.J. Williams, “A Simplified and Generalized Treatment of DES Related Ciphers”, Cryptologia 39:1, 3–24 (2015).

[4] Business Software Alliance, “Shadow Market: 2011 BSA Global Software Piracy Study”, ninth edition.
[5] C. Collberg, C. Thomborson, “Watermarking, Tamper-Proofing and Obfuscation - Tools for Software Protection”, Software Engineering, IEEE Transactions on Vol. 28, Issue 8 (2002) 735-746.
[6] I.B. Damgård, “A Design Principle for Hash Functions”, Lecture Notes in Computer Science Vol. 435, 416-427 (1989).
[7] I.B. Damgård, “ Collision free hash functions and public key signature schemes”, Advances in Cryptology, Proc. Eurocrypt’87 (1988), 203–216.
[8] I.B. Damgård, “The application of claw free functions in cryptography”, PhD Thesis, Aarhus University, Mathematical Institute, (1988).
[9] “ Data Encryption Standard, FIPS-Pub.46. National Bureau of Standards, U.S. Department of Commerce, Washington D.C., (1977).
[10] H. Feistel, “Cryptography and Computer Privacy”, Scientific American, Vol. 228 (1973), 15–23.
[11] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, “Fast and Vulnerable: A Story of Telematic Failures”, Proceedings of Workshop On Offensive Technologies (WOOT), Washington, D.C, (2015).
[12] J. Gyllenskog, “Inside the Smartdongle USB Security Key”, Dr. Dobb’s: The world of software development, (October 01 2005). URL: http://www.drdobbs.com/inside-the-smartdongle-usb-security-key/184406281
[13] J. Gyllenskog, “How DDJ Readers Helped Us Improve Our Product”, Dr. Dobb’s: The world of software development, (May 30 2008). URL: http://www.drdobbs.com/security/how-ddj-readers-helped-us-improve-our-pr/208401170
[14] D. Knuth, “The Art of Computer Programming”, Volume 2: Seminumerical Algorithms, Second Edition (1981).
[15] D.H. Lehmer, “Mathematical Methods in Large-scale Computing Units”, Proceedings of the Second Symposium on Large-Scale Digital Calculating Machinery, (1951)
[16] R. Merkle, “Secrecy, Authentication, and Public Key Systems”, UMI Research Press, (1979).
[17] R. Merkle, “One way hash functions and DES”, Advances in Cryptology, Proc. Crypto’89, LNCS 435, G. Brassard, Ed., Springer-Verlag (1990), 428–446.
[18] R. Merkle, “A Certified Digital Signature”, Lecture Notes in Computer Science Vol. 435 (1989), 218–238.
[19] S. Demetriou et al., “What’s in Your Dongle and Bank Account? Mandatory and Discretionary Protection of Android External Resources”, Network and Distributed System Security Proceedings (2015).
[20] U. Piazzalunga, P. Salvaneschi, F. Balducci, P. Jacomuzzi, and C. Moroncelli, “Security Strength Measurement for Dongle-Protected Software”, IEEE Security and Privacy 5, 6 (2007), 32-40.
[21] M.O. Rabin, “Digitalized signatures, Foundations of Secure Computation”, R. Lipton and R. DeMillo, Eds., Academic Press, New York (1978), 155–166.
[22] W.E. Thomson, “A Modeled Congruence Method of Generating Pseudorandom Numbers”, Computer Journal (1958).
[23] E. F. Schaefer, “ A Simplified Data Encryption Standard Algorithm, Cryptologia”, Vol. 20 (1996), 77–84.
[24] W. Trappe and L. C. Washington, “Introduction to Cryptography with Coding Theory”, Pearson Education, (2006).
[25] R. C. Phan, “Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students”, Cryptologia, Vol. 26 (2002), 283–306.

NOTES

[1]http://www.google.com/patents/US4168396?dq=4168396.
[2]This attack is based on using “snippets” mentioned at the end of the second paragraph of [13]. The 900 million years estimate reported in the fourth paragraph of [13] is based on an inefficient exhaustive search.
[3]We say more about $f_{A,n}$ in this section.