

Tame Transformation Signatures With Topsy-Turvy Hashes

Jiun-Ming Chen
Purdue University
W. Lafayette, Indiana, U.S.

Bo-Yin Yang
Tamkang University
Tamsui, Taiwan

`jmchen@math.purdue.edu`

`moscito@math.tku.edu.tw`

June 15, 2002

Abstract

We introduce the new $\text{GF}(2^8)$ -based digital signature scheme TTS (Tame Transformation Signatures). TTS is a consequence of the public-key cryptosystem TTM (Tame Transformation Method) and shares many of its superior properties, resulting in low signature delays, fast verification and high complexity. The commercial applications of TTS is protected under the patent of TTM. TTS can either be blended with currently fashionable hash functions (such as MD5 and SHA-1) or its own related hash function TTH (Topsy-Turvy Hash). We describe the principles and implementation of TTS and TTH, and analyze their properties – both in absolute and comparatively to alternative schemes.

1 Introduction

Secure authorization and authentication of information have been important and imminent problems in this age of the Internet. Identity fraud and sometimes outright theft runs rampant and many solutions have been proposed to rein in these beasts. Most involve some form of digital signatures and hash functions, hence faster and more secure hashes and digital signature schemes will be of great service in many ways.

In the course of this article, we will use the principles behind TTM (Tame Transformation Method, [11]) to derive a new digital signature scheme TTS (Tame Transformation Signatures). We also introduce a new related hash function TTH (Topsy-Turvy Hash). TTM, TTS and TTH all work on a finite field and have very similar designs. Due to their common ancestry, they share many properties including high complexity (security), ease of implementation and good execution speed. The following is a summary of the remaining sections:

Sec. 2 A brief recap of how Tame Automorphisms are used to construct the current TTM cryptosystem and the basic properties of TTM.

Sec. 3 The basic ideas behind Tame Transformation Signatures (TTS).

Secs. 4 and 5 Two simple but practical implementations of TTS.

Secs. 6 to 9 Analyzing properties of TTS, and comparison to alternatives.

Sec. 10 Describing a new related hashing scheme TTH and its features.

2 Tame Automorphism and TTM

A *Tame Automorphism* $\phi : \mathbf{x}(\in K^n) \mapsto \mathbf{y}(\in K^n)$ is usually given as a set of relations (where each q_i is a polynomial, and the subscripts can be permuted):

$$\begin{aligned} y_1 &= x_1; \\ y_2 &= x_2 + q_2(x_1); \\ y_3 &= x_3 + q_3(x_1, x_2); \\ &\vdots \\ y_i &= x_i + q_i(x_1, x_2, \dots, x_{i-1}); \\ &\vdots \\ y_n &= x_n + q_n(x_1, x_2, \dots, x_{n-1}). \end{aligned}$$

Tame Automorphisms had been first researched in algebraic geometry, but its use was first proposed by T. Moh in the context of public-key cryptography infrastructure ([11]). They possess the desirable property that

1. A preimage $\mathbf{x} = \phi^{-1}(\mathbf{y})$ can be computed very quickly by computing (solving for) each component serially, but:
2. an explicit polynomial form for ϕ^{-1} will be very hard to write out in full, being of very high degree with many, many terms:

$$\begin{aligned} x_1 &= y_1; \\ x_2 &= y_2 - q_2(x_1); \\ x_3 &= y_3 - q_3(x_1, x_2); \\ &= y_3 - q_3(y_1, y_2 - q_2(y_1)); \\ x_4 &= y_4 - q_4(x_1, x_2, x_3) \\ &= y_4 - q_4(y_1, y_2 - q_2(y_1), y_3 - q_3(y_1, y_2 - q_2(y_1))); \\ &\vdots \\ x_n &= y_n - q_n(x_1, x_2, \dots, x_{n-1}); \\ &= y_n - q_n(y_1, y_2 - q_2(y_1), \dots, y_{n-1} - q_{n-1}(\dots)). \end{aligned}$$

When TTM was first proposed it had an LTL (linear-tame-linear) form, with $K = \text{GF}(2^8)$, $\mathbf{y} = L_2 \circ T \circ L_1(\mathbf{x})$ (where \circ denotes composition, i.e. substitution). L_1 and L_2 are affine (linear) and T is tame and homogeneous quadratics for each q_i , and an expansion rate of 1 during encryption. This is susceptible to attacks by P. Montgomery and A. Sathaye (both unpublished) due to the fact that the first coordinate in the tame portion is fixed (as is practically the second coordinate, since q_2 is essentially constricted to $x \mapsto x^2$).

To ameliorate this flaw, current implementations of TTM (as in [11]) use an LTTL (linear-tame-tame-linear) form $\mathbf{y} = \phi_4 \circ \phi_3 \circ \phi_2 \circ \phi_1(\mathbf{x})$. Although

in principle, TTM may be of the form $LT \cdots TL$ with one or more T. In this form the components ϕ_1 and ϕ_4 are affine (linear), **but ϕ_2 is from K^n to K^r with $r > n$. It is really a Tame Automorphism in K^n applied after the canonical embedding $K^n \rightarrow K^r$ (i.e. pad x_{n+1}, \dots, x_r with zero's).** Again all displacements q_i are homogeneous and quadratic.

The major deviation concerns $\phi_3 : \mathbf{x} \mapsto \mathbf{y}$, which is a specially constructed Tame Automorphism $K^r \rightarrow K^r$ with this form:

$$\begin{aligned} y_1 &= x_1 + p_1(x_2, x_3, \dots, x_r); \\ y_2 &= x_2 + p_2(x_3, x_4, \dots, x_r); \\ y_3 &= x_3; \\ y_4 &= x_4; \\ &\vdots \\ y_r &= x_r; \end{aligned}$$

such that the degrees of p_1 and p_2 are suitably large (eight, in current practice) but the composition $\phi_3 \circ \phi_2 : K^n \rightarrow K^r$ are quadratic in each component of the image. Thus, $\phi = \phi_4 \circ \phi_3 \circ \phi_2 \circ \phi_1$ looks like a generic quadratic with r degree-2 equations of n variables each, and is given in the composite form as the public key. To achieve this desirable form, we need r to be substantially larger than n . Tradeoffs must be made between encryption time ($\propto (nr)$, or proportional to the square of encryption blocksize and the expansion rate) and safety. E.g. one of the current implementations of TTM uses $n = 16$ and $r = 48$ (multiples of 8 being easier on the programmer with computer architectures as they are).

The private key is the collection of all information built into each of the ϕ_i modified in such a way to maximize decryption speed.

Improved in the manner as described above, TTM is secure and speedy to the point where it can be used on its own and not in conjunction with a symmetric cipher. In more traditional PKI, a public key cryptosystem is only used to exchange the session key, and such is the case for the well-known SSH (Secure Shell) and PGP (Pretty Good Privacy) protocols. In its current form, TTM remains a “strong cryptosystem” under all known attacks (see Sec. 8). One can refer to [12]-[15] for more details.

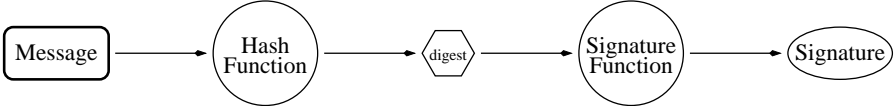
In retrospect, one might marvel at some of the similarities in the design of AES (Rijndael) and TTM/TTS. Both linear and non-linear operations are included. The linear operations have a diffusive effect, quickly mixing all components and masking underlying algebraic relations; but the non-linear operations are necessary to disrupt the structure of linear mappings (otherwise the result of a mapping will be determined by the result on a basis of the space). AES uses the action of taking a multiplicative inverse as the non-linear operation and in TTM/TTS we use quadratic polynomial substitutions.

Further information on TTM can be found at <http://www.usdsi.com>. We do want to stress that *computations pertaining to Tame Automorphisms (as well as other tame-like mappings) can be greatly accelerated using SIMD (Single Instruction Multiple Data) operations*, available (say) via Motorola’s AltiVec technology. See [10] for one such implementation for TTM.

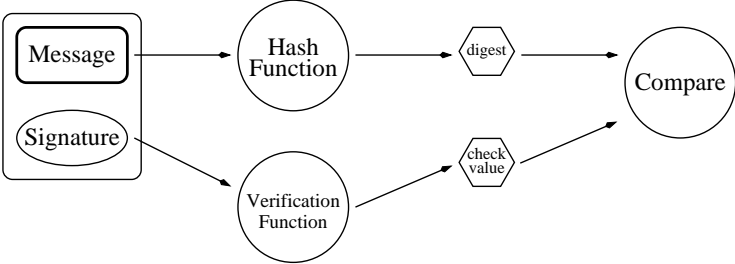
For an example of a “toy” component of TTM, please see [7].

3 Basic Idea behind TTS

To sign a message digitally, one takes its digest (hash) value according to some agreed hash function (in the well-known PGP protocol, this is SHA-1), then run that hash value through a signature function (for PGP, this is RSA-1024 or -2048). The result is the digital signature. During authentication, the recipient substitutes the signature into a “verification trap-door function” and compares the result with the message digest (using the same hash function). If they are the same, the message is presumed “clean”.



(a) Signing a message (with private key)



(b) Verifying a signature (with public key)

Figure 1: Digital Signature and Authentication Processes

We see that the use of Tame Automorphisms leads to strong cryptosystems with good properties. T. Moh discussed in his seminal paper [11] the theory on how to derive good digital signature schemes from using the same principles behind TTM (i.e. Tame Automorphisms), but our article is the first time that the details have been fleshed out and Tame Transformation proposed as the centerpiece of digital signature infrastructure without any mathematical cloud.

In a public-key cryptosystem one releases an injective function (the encryption map) whose trap-door inverse (the decryption map) is hard to find. In a digital signature scheme, the verification map released is an inverse to a hard-to-find injective trap-door (the signing map). Just fine for a symmetric cryptosystem like RSA, but the LTTL-form TTM encryption mapping is no longer a bijection (not surjective!) and cannot be used directly for digital signatures.

In TTS, we follow the theory in [11] and switch back to a LTL format for the mapping. The general idea is as follows: the message or hash value is padded out in a certain way before a tame transformation is applied. Again before and after the tame transformation there is an affine mapping portion.

Padding is a necessary evil because a simple LTL scheme will be relatively unsafe. If we pad zeros after each message, we obtain the scheme TTS-0; if we pad random numbers in front of each message, we get TTS-r. Via padding we maintain the kernel as (the identity plus) a homogeneous quadratic with the same nice properties as Tame Automorphisms.

4 TTS-0

TTS-0 is a basic implementation of digital signatures using Tame Automorphisms. The signature length is the (digest) hash length plus 4 bytes.

4.1 Signing a Message

In a basic version of TTS-0, the 24-byte signature $\mathbf{x} = \phi_1^{-1} \circ \phi_2^{-1} \circ \phi_3^{-1}(\mathbf{y})$ comes from a 20-byte digest value from the following component maps:

$$\begin{aligned}\phi_1 : K^{24} &\rightarrow K^{24}, & \mathbf{x} &\mapsto M_1\mathbf{x} + \mathbf{c}_1; \\ \phi_2 : K^{24} &\rightarrow K^{20}, & \mathbf{x} &\mapsto \text{Tame-like Map of } (\mathbf{x}); \\ \phi_3 : K^{20} &\rightarrow K^{20}, & \mathbf{x} &\mapsto M_3\mathbf{x} + \mathbf{c}_3.\end{aligned}$$

The Tame-like Transform portion ϕ_2 is given by the following equations:

$$\begin{aligned}y_0 &= x_0 + a_0 x_4 x_{20} + b_0 x_6 x_{21} + c_0 x_8 x_{22} + d_0 x_{10} x_{23}, \\ y_1 &= x_1 + a_1 x_5 x_{20} + b_1 x_7 x_{21} + c_1 x_9 x_{22} + d_1 x_{11} x_{23}, \\ y_2 &= x_2 + a_2 x_0 x_1 + b_2 x_{20} x_{21} + c_2 x_{12} x_{22} + d_2 x_{14} x_{23}, \\ y_3 &= x_3 + a_3 x_1 x_2 + b_3 x_{22} x_{23} + c_3 x_{13} x_{20} + d_3 x_{15} x_{21}, \\ y_4 &= x_4 + a_4 x_0 x_2 + b_4 x_1 x_3 + c_4 x_{16} x_{20} + d_4 x_{18} x_{22}, \\ y_5 &= x_5 + a_5 x_0 x_3 + b_5 x_1 x_4 + c_5 x_{17} x_{21} + d_5 x_{19} x_{23}, \\ y_6 &= x_6 + a_6 x_0 x_4 + b_6 x_1 x_5 + c_6 x_2 x_3 + d_6 x_{20} x_{22}, \\ y_7 &= x_7 + a_7 x_0 x_5 + b_7 x_1 x_6 + c_7 x_2 x_4 + d_7 x_{21} x_{23}, \\ y_8 &= x_8 + a_8 x_0 x_6 + b_8 x_1 x_7 + c_8 x_2 x_5 + d_8 x_3 x_4, \\ y_9 &= x_9 + a_9 x_0 x_7 + b_9 x_1 x_8 + c_9 x_2 x_6 + d_9 x_3 x_5, \\ y_{10} &= x_{10} + a_{10} x_2 x_9 + b_{10} x_3 x_8 + c_{10} x_4 x_7 + d_{10} x_5 x_6, \\ y_{11} &= x_{11} + a_{11} x_3 x_{10} + b_{11} x_4 x_9 + c_{11} x_5 x_8 + d_{11} x_6 x_7, \\ &\vdots & \quad \quad \quad \vdots \\ y_k &= x_k + a_k x_{k-8} x_{k-1} + b_k x_{k-7} x_{k-2} + c_k x_{k-6} x_{k-3} + d_k x_{k-5} x_{k-4}, \\ &\vdots & \quad \quad \quad \vdots \\ y_{19} &= x_{19} + a_{19} x_{11} x_{18} + b_{19} x_{12} x_{17} + c_{19} x_{13} x_{16} + d_{19} x_{14} x_{15}.\end{aligned}$$

It should be readily visible from the above formulas that

- If $x_{20} = x_{21} = x_{22} = x_{23} = 0$ then this is a Tame Automorphism; and
- the first ten equations are pretty much constructed *ad hoc post hoc* to provide for distinct quadratic square-free terms for all equations; but

$$\begin{aligned}
y_7 &= x_7 = \text{Uniform Random Variable in } K; \\
y_8 &= x_8 + a_8 x_0 x_7 + b_8 x_1 x_6 + c_8 x_2 x_5 + d_8 x_3 x_4; \\
y_9 &= x_9 + a_9 x_1 x_8 + b_9 x_2 x_7 + c_9 x_3 x_6 + d_9 x_4 x_5; \\
&\vdots \quad \quad \quad \vdots \\
y_k &= x_k + a_k x_{k-8} x_{k-1} + b_k x_{k-7} x_{k-2} + c_k x_{k-6} x_{k-3} + d_k x_{k-5} x_{k-4}, \\
&\vdots \quad \quad \quad \vdots \\
y_{31} &= x_{31} + a_{31} x_{23} x_{30} + b_{31} x_{24} x_{29} + c_{31} x_{25} x_{28} + d_{31} x_{26} x_{27};
\end{aligned}$$

where $\mathbf{y} = (y_8, y_9, \dots, y_{31}) \in K^{24}$ (note subscripts), and $\mathbf{x} = (x_0, x_1, \dots, x_{31}) \in \phi_2^{-1}(\mathbf{y}) \subset K^{32}$ will be constructed during the signing process by solving the above equations. The signing portion or private key is given by the information in each part of the composite $\phi_1^{-1} \circ \phi_2^{-1} \circ \phi_3^{-1}$, and during the process each step is evaluated separately.

As we would mention below, it's possible to take certain shortcuts in the signing process (see 9.1).

5.2 Verifying a Message

Verifying under TTS-r is nearly identical as under TTS-0. The verification mapping or public key is given by the composite of $\phi_3 \circ \phi_2 \circ \phi_1$. This also evaluates to a generic set of quadratic relations

$$y_i = \mathbf{x}^T A_i \mathbf{x} + \mathbf{b}_i^T \mathbf{x}, \quad i = 8, \dots, 31.$$

The matrices A_i and the vectors \mathbf{b}_i therein are then recorded as the public key. We should mention *again* that normally there would be a constant term for each y_i except that we adjust \mathbf{c}_3 above to zero out them all. Key sizes (mildly larger than when using TTS-0 with the same hash function) can be found in Table 1.

	TTS-0			TTS-r		
equations	16	20	24	16	20	24
variables	20	24	28	24	28	32
message (bits)	128	160	192	128	160	192
signature (bits)	160	192	224	192	224	256
linear terms	20	24	28	24	28	32
quad. terms	210	300	406	300	406	528
terms per eq.	230	324	434	324	434	560
public key #par	3680	6480	10416	5184	8680	13440
linear-1 #coeff	420	600	812	600	812	1056
TAME #coeff	64	80	96	64	80	96
linear-3 #coeff	272	420	600	272	420	600
private key #par	756	1100	1508	936	1312	1752

Table 1: Comparison of basic key lengths for TTS-0 and TTS-r

TTS-r is slightly unusual in the sense that the same message may not result in the same signature (in practice not a problem).

6 Flexibility

Both TTS variants can be easily adapted to any size of hash value. For example, in TTS-r, a hash of 160 bits or twenty bytes (using a $K^{20} \rightarrow K^{28}$ Tame Automorphism) would be chosen for compatibility with SHA-1, with a 224-bit (28 bytes) signature; a similar construction for MD5 with a 128-bit hash value would generate a 192-bit signature in TTS-r (and 160 bits in TTS-0). If necessary, “trampolines” (data created to be programs) corresponding to other hash sizes can be created on the fly in response to the user’s needs.

TTS can always be adapted by adding more square-free quadratic terms for higher level of security. This action will not affect the speed of verification, just increase the size of the private key and slightly slow the signing speed.

7 A comparative study

TTS (and TTH) like current implementations of TTM work with the affine spaces over the finite field $\text{GF}(2^8)$. We venture our humble opinion that small working sizes associated with finite fields can be easier implemented effectively than alternatives with very large working sizes, such as the astronomically big groups used by RSA, ECC, ElGamal and their relatives. A point in support is that AES (see [1] and [5]), favorite symmetric cryptosystem *du jour* based on the Rijndael block cipher, is also based on computations over the very same finite field $\text{GF}(2^8)$. The choice of $\text{GF}(2^8)$ is natural as not only is it convenient computationwise, it allows natural subfields of $\text{GF}(2^4)$ and $\text{GF}(2^2)$, making for easier practical implementation with reasonably-sized keys.

7.1 Comparison to traditional PKI

Most extant PKI (Public Key Infrastructure) are based on RSA. A main drawback of RSA signatures is its large signature sizes (usually 1024 or 2048 bits, much larger than finite-field-based ones). RSA schemes *do* have relatively smaller keys, however. Signing a message under RSA is again significantly slower than under TTS, due to the underlying structures. Key generation is again faster under TTS than under RSA. All in all, there are distinctive advantages to use TTS rather than RSA. The superiority is more pronounced against (say) the even slower DSA/DSS (based on discrete logarithms).

7.2 Comparison against other multivariate schemes

In any multivariate quadratic finite-field based schemes (to which TTS belongs) like SFLASH ([19]), to verify a message is much like encrypting a message in the corresponding Public Key Cryptosystem – that is, verification in SFLASH is also substitutions into a set of quadratic polynomials just like TTS, so uses comparable time and resources. QUARTZ ([18]), which also has polynomial substitution for this step, performs similarly here.

However, signing a message is really like decrypting a message under the analogous cryptosystem, we see substantial performance differences (Table 2). The structure of TTS is such that decrypting is again a substitution process and hence it is easy to code and quick to run. SFLASH (based on C^{*--}) is more

troublesome in this aspect. The signing procedure for QUARTZ, equivalent to the decrypting process of HFE from which it is derived, is based on solving equations in a largish finite field, and as such is hard to code for and snail-paced in comparison. One can see from the data in Table 2 that TTS shines against (at least) these rival schemes for digital signatures.

	QUARTZ	SFLASH	TTS-0	TTS-r
Length of signature (bits)	128	259	192	224
Length of public key (kB)	71	15.4	6.4	8.6
Length of private key (kB)	3	2.4	1.1	1.3
Generating key pair (sec)	4 (avg.)	≈ 1	$\approx 10^{-2}$	$\approx 10^{-2}$
Signing a message (sec)	10 (avg.)	2.7×10^{-3}	$\approx 10^{-4}$	$\approx 10^{-4}$
Verify a signature (sec)	$\approx 10^{-3}$	8×10^{-4}	$\approx 10^{-3}$	$\approx 10^{-3}$

Table 2: Comparison of signature schemes (times on a Pentium III/500).

The underlying structure is also why TTS generates keys faster than SFLASH or QUARTZ. One might also point out, of course, that the $GF(2^7)$ used in SFLASH instead of $GF(2^8)$ makes it impossible to exploit the existence of intermediate field extensions. In Sec. 9 we can see how the existence of compatible subfields of $GF(2^8)$ can help with TTS.

Lastly, there is no chance that an attempt to sign a message will fail; as incredible as it sounds (and as small as this probability is), this possibility actually exists in QUARTZ.

8 Brief Cryptanalysis

The problem of solving a set of quadratic equations is NP-complete ([6], [9]), hence unless there is an inherent flaw in TTS, no attack can do much more damage than brute force. Of course, given the kinship between TTM and TTS, most attacks that applies to an implementation of TTM may be surmised to apply to TTS, and some notables have made notable attempts:

1. Jacques Patarin in [16] and later [17] gave and used algorithms for solving IP (Isomorphism of Polynomials) Problem: given sets of two quadratic relations (mappings $\mathbf{x} \in K^m \mapsto \mathbf{y} \in K^n$ with $y_j = f_j(x_1, \dots, x_m)$, $j = 1 \dots n$.) P and Q , find affine (linear) mappings L_1 and L_2 such that $Q = L_1 \circ P \circ L_2$. (Reduction to) IP is ineffective against TTM because (see [14]) Patarin’s algorithm requires explicit knowledge of P and Q , but all current production-quality Tame Automorphism based methods include many user-determined terms in the kernel mappings.
2. Kipnis and Shamir invented the **relinearization** improvement to the usual linearization approach solving sets of high-order equations, based on the simple fact that

$$(ab)(cd) = (ac)(bd) = (ad)(bc),$$

in any field. It does ameliorate to some extent the problem of too many extraneous solutions and is used against HFE in [8]. However as is shown in

[12] such an attack does not decrease the amount of computations needed for solving many implementations of TTM to a manageable level.

3. In [4] we see that experiments have been performed by N. Courtois., A. Klimov, A. Shamir, and J. Patarin with new methods “XL” and “FXL” against so-called “overdetermined cryptosystems”. Since this means more equations than independent variables and include no current systems other than Tame Automorphism based systems, one might be justified in concluding that someone had (some version of) TTM in his sights. However, it is illustrated in [13] that such attempts are futile according to the century-old mathematical texts of Hilbert and Serre, because for XL to have any efficacy the solution set at ∞ must be either empty or 0-dimensional.
4. In [3] N. Courtois, and J. Goubin asserted that the **MinRank attack** is effective against TTM. Unfortunately the paper was so full of inaccuracies as pointed out in [15], that it is hard to see how the Courtois-Goubin attack can be mathematically sound.

In each polynomial of the kernel Tame Transformation for any TTS scheme is four distinct square-free quadratic terms, none of which ever being replicated in another polynomial. Any linear combinations will not result in elimination of any terms. 4 square-free terms are considered plenty by current authorities; should there be new attacks that can exploit the smallness of number of terms in each polynomial, a few new terms can be appended without substantial penalty.

As in the case for TTM (see [11] for details), any cracking attempt will not do much better than a brute force search and all reasonable implementations of TTS has an effective complexity above 2^{80} .

9 Key Size Reduction

As can be seen above, TTS has significantly larger keys than those in RSA, although they are still smaller than say in QUARTZ. We can make it smaller and more manageable with one of several restrictions on the mappings involved.

9.1 Public Key

Reducing the size of public keys in TTS is essentially the same as for TTM. Fix a 4-bit subfield $K' = \text{GF}(2^4)$ that is compatible (has a compatible multiplication table) with our $\text{GF}(2^8)$, and use elements from K' for all the quadratic terms in the Tame Transformation and all matrix elements in the affine mappings. This effectively cut the public key size in half and can also nearly double the execution speed with suitable programming exercises.

Of course, the time needed to crack the scheme also goes down, but the analysis show that the complexity is still easily beyond the reach of crackers in the foreseeable future.

9.2 Private Keys

There are various ways to decrease size of private keys. Of course, using a compatible $K' = \text{GF}(2^4)$ will also decrease the private key size, but we can cut down further on this size by either of these tricks:

1. Using 1-bit entries for the linear-part square matrices. For TTS-r with 20-byte hashes and 28-byte signature, we would instead of $28^2 + 20^2$ bytes in the private keys have 1/8 times that many bytes for a total of 148.

This key reduction trick enables the programmer to avoid costly table lookups and to use a variety of common SIMD techniques to do several operations in parallel, and some of these methods does not even require the CPU to have SIMD instructions!

2. Using the composition of a diagonal matrix L , a complement $J - P$ of a permutation matrix (exactly one entry out of every row and column being 0 with the others being 1) and another diagonal matrix R for the matrices in the linear parts. In other words, $M_1 = L_1(J - P_1)R_1$, and $M_3 = L_3(J - P_3)R_3$ for the matrices of the linear portions. Noting that with characteristic-2 fields, $J - P$ is unitary for permutation matrices of even order, $M_1^{-1} = R_1^{-1}(J - P_1)^T L_1^{-1}$ and $M_3^{-1} = R_3^{-1}(J - P_3)^T L_3^{-1}$. Using 20-byte hashes with TTS-r (i.e. 28-byte signatures), we would have instead of $28^2 + 20^2 = 1184$ bytes in private keys a rather smaller $(28 + 28 + 28) + (20 + 20 + 20) = 144$ bytes in private keys.

This trick cuts the running time from $O(s^2)$ to $O(s)$, where s is the number of bytes in the signature.

Both of these simple tricks would cut the private key sizes for TTS-r (20 → 28) down to < 300 (there are parameters in constant parts and the Tame Automorphism part), comparable to RSA.

10 A Fast Flexible Hash Function

We quote the excellent criteria for hash functions in [20]:

The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be useful for message authentication, a hash function H must have the following properties:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. $H(x)$ is relatively easy to compute for any given x , making both hardware and software implementations practical.
4. (**One-way property**) For any given code h , it is computationally infeasible to find x such that $H(x) = h$.
5. (**Weak collision resistance**) For any given block x , it is computationally infeasible to find $y \neq x$ such that $H(y) = H(x)$.
6. (**Strong collision resistance**) It is computationally infeasible to find a pair of blocks $x \neq y$ such that $H(x) = H(y)$.

The most commonly used hash functions include the 128-bit MD5 and the 160-bit SHA-1 (the default in PGP) and RIPEMD. Due to a birthday attack possibly reducing the complexity of building a plaintext with any given MD5 hash to 2^{64} , most authorities recommend at least 160-bit as a minimum safeguard.

One common message authentication algorithm (MAC) is based on a CBC (cipher block chaining) mode DES encryption operation: The data to be authenticated is padded appropriately and divided into 64-bit blocks D_1, D_2, \dots, D_n . Using the DES algorithm E with secret key K, the DAC (data authentication code) is computed as follows:

$$\begin{aligned} h_1 &= E_k(D_1), \\ h_2 &= E_k(D_2 \text{ xor } h_1), \\ h_3 &= E_k(D_3 \text{ xor } h_2), \\ &\vdots \\ h_n &= E_k(D_n \text{ xor } h_{n-1}). \end{aligned}$$

With the DAC being h_n or some leftmost portion thereof. The construction of our hash, which we will call TTH (Topsy-Turvy Hash), is quite similar:

1. Padding: one “0” bit and a number of “1” bits are appended to the message block so that the length is congruent to $8k - 64 \pmod{8k}$, where k is number of bytes in a hash value, usually 20 or 24. Then append the length (modulo 2^{64}) of the message as a long integer in little-endian format. Note that like MD5 *et al*, we always pad – even when the length is already $8k - 64 \pmod{8k}$ (i.e. pad another $8k$ -bits in that case).
2. Divide the resulting string into $8k$ -bit blocks D_1, \dots, D_n , then recursively construct the following:

$$\begin{aligned} h_1 &= T_\pi(D_1 \text{ xor } C), \\ h_2 &= T_\pi(D_2 \text{ xor } h_1 \text{ xor } C), \\ h_3 &= T_\pi(D_3 \text{ xor } h_2 \text{ xor } C), \\ &\vdots \\ h_n &= T_\pi(D_n \text{ xor } h_{n-1} \text{ xor } C); \end{aligned}$$

where C is a random k -byte vector and T_π a set of generic quadratic polynomials with its coefficients picked from the binary expansion of π (essentially a truly random bitstream). **The hash value is h_n .** There will be no structure, decomposition, or easy inversion and it is an NP-complete problem to solve such equations ([6], [9]).

Obviously, the *xor* with some essentially random but fixed string is to avoid certain types of man-in-the-middle attacks.

Why do we need yet another hash function when there are already so many others around? There are at least three such considerations:

1. The size of the hash value is easily adjustable. The basic approach would not be in danger of being discarded as the MD5 algorithm now seem to be. With TTH, we can simply shift to a higher number of bits.
2. The algorithm is simpler than SHA-1 or MD5, and easier to implement.
3. We can use “for free” to compute TTH the same subroutine used for other Tame-Automorphism-related substitutions.

This last point is exploited in production code – see the following segment in a library routine used for verification in TTS, encryption in TTM, and TTH (Note: `MT[(a<<8)|b]` performs a look up for `ab` in $GF(2^8)$, in the multiplication table `MT[65536]`):

```

/* N_eq    number of polynomials
   N_var    number of variables
   N_term   number of terms in quadratic,
            equal to (N_var+1)(N_var+2)/2 */

typedef unsigned char  uint1; /* treated as elements in GF(256) */
typedef unsigned int   uint4;
typedef uint1 QuadPoly[N_term]; /* linear coefficients followed by
                                quadratics, lexicographical order */
typedef QuadPoly PublicKey[N_eq];

inline uint4 shift(uint1 u) {
    return ((uint4) u) << 8;
}

/* evaluate each polyn y_i = (C_i + Xt U_i) X, C_i is a vector,
   U_i is upper-triangular, and Xt is the transpose of vector X */

void EvaluatePoly(const QuadPoly & coeff, const uint4 * var, uint1 & value){
    const uint1 * Cptr = coeff + N_term - 1; /* pointer quadratic terms */
    int row, col;
    uint4 sum;

    value = 0;
    for (col = N_var - 1; col >= 0; col--) {
        sum = coeff[col];
        for (row = col; row >= 0; row--, Cptr--)
            sum ^= MT[var[row]|*Cptr]);
        value ^= MT[var[col]|sum];
    }
}

void TransformBlock(const uint1 * InBlock, uint1 * OutBlock) {

    uint4 variable[N_var];
    int n;

    /* shift the values of variables for table look-up */
    for (n = N_var - 1; n >= 0; n--) variable[n] = shift(InBlock[n]);

    /* evaluate N_eq polynomials to get a block as output */
    for (n = N_eq - 1; n >= 0; n--)
        EvaluatePoly(PublicKey[n], variable, OutBlock[n]);
}

```

Acknowledgements

The authors would like to thank Prof. T.-T. Moh for his time, advice and encouragement, without which this paper would never have been written.

The last-named author would like to thank Profs. Kleitman and Stanley of the MIT Mathematics Department for their hospitality during the school year 2001-02 as well as the most capable Shirley and Camillo during his stay.

References

- [1] AES home page - <http://csrc.nist.gov/encryption/aes>
- [2] C.-Y. Chou, D.-J. Guan and J.-M. Chen, *A Systematic Construction of a Q_2^k -module in TTM*, Communications in Algebra, 30(2002), 551-562.
- [3] N. Courtois and J. Goubin, *Cryptanalysis of the TTM Cryptosystem*. pp. 44-57 in *Asiacrypt 2000, Lecture Notes in Computer Science 1976*, Springer-Verlag, 2000.
- [4] N. Courtois., A. Klimov, A. Shamir, and J. Patarin, *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations* pp. 392-407 in *Eurocrypt 2000, Lecture Notes in Computer Science 1807*, Springer-Verlag, 2000.
- [5] Joan Daemen and Vincent Rijmen, *The Design of Rijndael, AES - The Advanced Encryption Standard*. Springer-Verlag, 2002.
- [6] Michael Garey and David Johnson, *Computers and Intractability, a guide to the theory of NP-completeness*. Freeman, 1979.
- [7] Y. Hu, L. Wang, J. Chen, F. Lai, and C. Chou, *A Performance Report and Security Analysis of a fast TTM implementation*, preprint.
- [8] A. Kipnis and A. Shamir, *Crypanalysis of the HFE public key cryptosystem*, pp. 19-30 in *Crypto 1999, Lecture Notes in Computer Science 1666*, Springer-Verlag, 1999.
- [9] K. Manders and L. Adleman, *NP-complete decision problems for quadratic polynomials*, pp. 23-29 in *Conference record of the eighth annual ACM Symposium on Theory of Computing: papers presented at the Symposium, Hershey, Pennsylvania, May 3-5, 1976*, ACM Press, 1976.
- [10] B. Lucier, *Cryptography, Finite Fields, and Altivec*, preprint available at <http://www.altivec.org/articles/>
- [11] T. Moh, *A Public Key System with Signature and Master Key Functions*. Communications in Algebra, 27 (1999) 2207-2222.
- [12] T. Moh, *Relinearization and TTM*, Preprint 1999.
- [13] T. Moh, *On The Method of XL and Its Inefficiency Against TTM* in Cryptology ePrint Archive (2001/47).

- [14] T. Moh, *A Cryptanalysis of TTM in Multivariate Cryptography*, International Press, 2003.
- [15] T. Moh and J.-M. Chen, *On the Goubin-Courtois Attack on TTM* in Cryptology ePrint Archive (2001/72). Moh's papers also available at <http://www.usdsi.com> and <http://www.chnds.com>.
- [16] Jacques Patarin, *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms*, pp. 33-48 in *Eurocrypt 1996, Lecture Notes in Computer Science 1070*, Springer-Verlag, 1996.
- [17] J. Patarin, N. Courtois, and J. Goubin, *Improved Algorithms for Isomorphisms of Polynomials*, pp. 184-200 in *Eurocrypt 1998, Lecture Notes in Computer Science 1403*, Springer-Verlag, 1998.
- [18] J. Patarin, N. Courtois, and J. Goubin, *Quartz, 128-bit long digital signatures*. in *Cryptographers' Track RSA Conference 2001, Lecture Notes in Computer Science 2020*, Springer-Verlag 2001; updated version at <http://www.cosic.esat.kuleuven.ac.be/nessie/tweaks.html>
- [19] J. Patarin, N. Courtois, and J. Goubin, *Flash, a fast multivariate signature algorithm*. in *Cryptographers' Track RSA Conference 2001, Lecture Notes in Computer Science 2020*, Springer-Verlag 2001; updated version at <http://www.cosic.esat.kuleuven.ac.be/nessie/tweaks.html>
- [20] W. Stallings, *Cryptography and Network Security : Principles and Practice*, 2nd ed. Prentice Hall, 1998.