# Tame Transformation Signatures With Topsy-Turvy Hashes

Jiun-Ming Chen[1] Bo-Yin Yang[2] Bor-Yuan Peng[3]

[1] Purdue University, W. Lafayette, Indiana, USA. <jmchen@math.purdue.edu>
[2] Tamkang University, Tamsui, Taiwan. <by@moscito.org>
[3] National Taiwan University, Taipei, Taiwan. <r0921022@ee.ntu.edu.tw>

## Abstract

We introduce the new $\mathrm{GF}(2^8)$-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.

**keywords: finite field, tame automorphism, digital signature, TTM, TTS, TTH**

## 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, [14]) 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.** Describing the basic ideas behind TTS (Tame Transformation Signatures).

**Secs. 4.–5.** Two practical TTS implementations.

**Secs. 6.–9.** Qualitatitve and relative analysis of TTS.

**Sec. 10.** A new hash function TTH and its features.

## 2. Tame Automorphisms to 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):

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

Tame Automorphisms had been first researched in algebraic geometry, but its use was first proposed by T. Moh for public-key cryptography infrastructure ([14]). They possess the desirable property that

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

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

When TTM was first proposed it had an LTL (linear-tame-linear) form, with $K = \mathrm{GF}(2^8)$, $\mathbf{y} = L_2 \circ T \circ L_1(\mathbf{x})$ (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 ([14]) use an LTTL (linear-tame-tame-linear) form[1] $\mathbf{y} = \phi_4 \circ \phi_3 \circ \phi_2 \circ \phi_1(\mathbf{x})$. The components $\phi_1$ and $\phi_4$ are affine (linear), **but $\phi_2$ is from $K^n$ to $K^r$ with $n < r$. It is really a Tame Automorphism in $K^r$ applied after the canonical embedding $K^n \to K^r$ (i.e. pad $x_{n+1}, \ldots, x_r$ 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 $K^r \to K^r$ with this form (where $1 < s < r$):

$$\begin{aligned} y_1 &= x_1 + p_1(x_2, x_3, \ldots, x_r); \\ y_2 &= x_2 + p_2(x_3, x_4, \ldots, x_r); \\ \vdots &= \vdots \\ y_s &= x_s + p_s(x_{s+1}, \ldots, x_r); \\ y_{s+1} &= x_{s+1}; \\ \vdots &= \vdots \\ y_r &= x_r; \end{aligned}$$

such that the degrees of the polynomials $p_j$'s are suitably large[2] but the composition $\phi_3 \circ \phi_2 : K^n \to K^r$ 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 get 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 block-size and the expansion rate) and safety. E.g. a current implementations of TTM uses $n = 20$ and $r = 48$, multiples of 4 being easier on the programmer (on current computer architectures).

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

Improved as 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. 9. and [15]-[18] for more details).

In retrospect, one might marvel at some of the similarities in the design of AES(Rijndael) and TTM/TTS. Both work on $\mathrm{GF}(2^8)$ and include both linear and non-linear operations. 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.

[1] In principle, the TTM encryption map can be $LT \cdots TL$ with one or more $T$; more $T$'s can be added as security dictates; in practice this is seldom necessary.

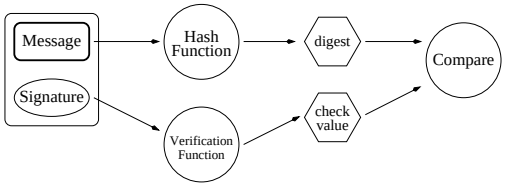
[2] $\deg p_1 = \deg p_2 = 6$, the other $p_j$'s are a handful of cubics.

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 [13] for one such implementation for TTM.

## 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

T. Moh's seminal paper [14] sketched how to derive digital signature schemes using the same principles behind TTM (i.e. Tame Automorphisms) known to yield strong cryptosystems with good properties, 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 [14] 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:

$$\begin{aligned}
&\phi_1 : K^{24} \to K^{24}, \quad \mathbf{x} \mapsto M_1\mathbf{x} + \mathbf{c_1};\\
&\phi_2 : K^{24} \to K^{20}, \quad \mathbf{x} \mapsto \text{Tame-like Map of } (\mathbf{x});\\
&\phi_3 : K^{20} \to K^{20}, \quad \mathbf{x} \mapsto M_3\mathbf{x} + \mathbf{c_3}.
\end{aligned}$$

The Tame-like Transform portion $\phi_2$ is given by:

$$\begin{aligned}
y_0 &= x_0 + a_0x_4x_{20} + b_0x_6x_{21} + c_0x_8x_{22} + d_0x_{10}x_{23},\\
y_1 &= x_1 + a_1x_5x_{20} + b_1x_7x_{21} + c_1x_9x_{22} + d_1x_{11}x_{23},\\
y_2 &= x_2 + a_2x_0x_1 + b_2x_{20}x_{21} + c_2x_{12}x_{22} + d_2x_{14}x_{23},\\
y_3 &= x_3 + a_3x_1x_2 + b_3x_{22}x_{23} + c_3x_{13}x_{20} + d_3x_{15}x_{21},\\
y_4 &= x_4 + a_4x_0x_2 + b_4x_1x_3 + c_4x_{16}x_{20} + d_4x_{18}x_{22},\\
y_5 &= x_5 + a_5x_0x_3 + b_5x_1x_4 + c_5x_{17}x_{21} + d_5x_{19}x_{23},\\
y_6 &= x_6 + a_6x_0x_4 + b_6x_1x_5 + c_6x_2x_3 + d_6x_{20}x_{22},\\
y_7 &= x_7 + a_7x_0x_5 + b_7x_1x_6 + c_7x_2x_4 + d_7x_{21}x_{23},\\
y_8 &= x_8 + a_8x_0x_6 + b_8x_1x_7 + c_8x_2x_5 + d_8x_3x_4,\\
y_9 &= x_9 + a_9x_0x_7 + b_9x_1x_8 + c_9x_2x_6 + d_9x_3x_5,\\
y_{10} &= x_{10} + a_{10}x_2x_9 + b_{10}x_3x_8 + c_{10}x_4x_7 + d_{10}x_5x_6,\\
y_{11} &= x_{11} + a_{11}x_3x_{10} + b_{11}x_4x_9 + c_{11}x_5x_8 + d_{11}x_6x_7,\\
&\vdots \quad \vdots\ \vdots\\
y_k &= x_k + a_kx_{k-8}x_{k-1} + b_kx_{k-7}x_{k-2}\\
&\qquad + c_kx_{k-6}x_{k-3} + d_kx_{k-5}x_{k-4},\\
&\vdots \quad \vdots\ \vdots\\
y_{19} &= x_{19} + a_{19}x_{11}x_{18} + b_{19}x_{12}x_{17}\\
&\qquad + c_{19}x_{13}x_{16} + d_{19}x_{14}x_{15}.
\end{aligned}$$

It is 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
- the second half of these equations are formed in a regular manner; and
- **can use any number of equations above 10**.

Example: For a 16-byte message digest (as in MD5) and 20-byte signature, the number of equations in the regular pattern is decreased by 4 and $(x_{16}, x_{17}, x_{18}, x_{19})$ replaces $(x_{20}, x_{21}, x_{22}, x_{23})$.

Each of the coefficients $a_i$, $b_i$, $c_i$, $d_i$ can be chosen freely as long as it is non-zero. The linear mappings can also be chosen arbitrarily except that we pick (compute) $\mathbf{c}_3$ of the affine (linear) mappings so that the resulting mapping $\phi_3 \circ \phi_2 \circ \phi_1$ has no constant part. Essentially, the freely chosen coefficients are stored in a form for fast execution of the signing code as the private key.

We can now compute $\phi_2^{-1}$ (i.e. solve for $\mathbf{x}$) with the conditions $x_{20} = x_{21} = x_{22} = x_{23} = 0$. Clearly, given any digest $\mathbf{y}$, we can compute the signature $\mathbf{x}$ quickly under these constraints.

### 4.2 Verifying a Message

Given a message $M$ and its digital signature $\mathbf{x}$, we hash its digest value $\mathbf{y} = H(M)$ and check to see if $\mathbf{y} = \phi_3 \circ \phi_2 \circ \phi_1(\mathbf{x})$. If yes, the message is presumed genuine else it is rejected as likely fake. The composite $\mathbf{x} \mapsto \mathbf{y}$ mapping is a generic quadratic set of polynomials without any constant terms.

$$y_i = \mathbf{x}^T A_i\, \mathbf{x} + \mathbf{b}_i^T\mathbf{x}, \quad i = 0, \ldots 19.$$

The upper triangular matrices $A_i$ and vectors $\mathbf{b}_i$ (suitably modified for speed) are stored as the public key. (See Table 1 for the sizes of keys).

## 5. TTS-r

This implementation utilizes 8 random elements from $K = \mathrm{GF}(2^8)$ and the signature will be 8 bytes longer than the hash value. 64 bits from a sufficiently random bitstream is needed; without dedicated hardware, the best source is Operating System level entropy collection based on I/O latency, as in the pseudodevice files `/dev/random` or `/dev/urandom` in various free Unices.

TTS-r is slightly unusual in that the same message may not result in the same signature. In practice this is not a problem.

### 5.1 Signing a Message

Without loss of generality, take $\mathbf{y}$ to represent a 24-byte (192 bits) hash value. We obtain the signature $\mathbf{x} = \phi_1^{-1} \circ \phi_2^{-1} \circ \phi_3^{-1}(\mathbf{y})$, an 32-byte string, with the following component maps:

$$\begin{aligned}
&\phi_1 : K^{32} \to K^{32}, \quad \mathbf{x} \mapsto M_1\mathbf{x} + \mathbf{c_1};\\
&\phi_2 : K^{32} \to K^{24}, \quad \mathbf{x} \mapsto \text{Tame-Transform of } (\mathbf{x});\\
&\phi_3 : K^{24} \to K^{24}, \quad \mathbf{x} \mapsto M_3\mathbf{x} + \mathbf{c_3}.
\end{aligned}$$

The Tame Transform portion $\phi_2$ is given by:

$$\begin{aligned}
y_0 &= x_0 = \text{Uniform Random Variable in } K;\\
&\vdots \quad \vdots\ \vdots\\
y_7 &= x_7 = \text{Uniform Random Variable in } K;
\end{aligned}$$

$$
\begin{aligned}
y_8 &= x_8 + a_8x_0x_7 + b_8x_1x_6 + c_8x_2x_5 + d_8x_3x_4;\\
y_9 &= x_9 + a_9x_1x_8 + b_9x_2x_7 + c_9x_3x_6 + d_9x_4x_5;\\
&\vdots \quad \vdots \quad \vdots\\
y_k &= x_k + a_kx_{k-8}x_{k-1} + b_kx_{k-7}x_{k-2}\\
&\qquad + c_kx_{k-6}x_{k-3} + d_kx_{k-5}x_{k-4},\\
&\vdots \quad \vdots\ \vdots\\
y_{31} &= x_{31} + a_{31}x_{23}x_{30} + b_{31}x_{24}x_{29}\\
&\qquad + c_{31}x_{25}x_{28} + d_{31}x_{26}x_{27};
\end{aligned}
$$

where $\mathbf{y} = (y_8, y_9, \ldots, y_{31}) \in K^{24}$ (note subscripts), and $\mathbf{x} = (x_0, x_1, \ldots, x_{31}) \in \phi_2^{-1}(\mathbf{y}) \subset K^{32}$ 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 will be shown below (see Sec. 8.), it's possible to take certain shortcuts in the signing process.**

## 5.2 Verifying a Message

Verifying in TTS-r is nearly identical as in 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

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

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. Basic key lengths for TTS-0 and TTS-r

# 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 $K^{28} \to K^{20}$ 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 be adapted for higher level of security by adding more square-free quadratic terms. This 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 $\mathrm{GF}(2^8)$. 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. AES (see [1] and [7]), favorite symmetric cryptosystem *du jour* based on the Rijndael block cipher, is also based on computations over the very same finite field $\mathrm{GF}(2^8)$. $\mathrm{GF}(2^8)$ is a natural choice for being convenient computationwise and allowing compatible subfields of $\mathrm{GF}(2^4)$ and $\mathrm{GF}(2^2)$ (thus 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. RSA signatures are much larger (usu. 1024 or 2048 bits, much larger than in finite-field-based schemes). Signing a message under RSA is also significantly slower than under TTS, as is generating a key pair. 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). **The length of keys is a common bugbear of multivariate quadratic cryptography. In Sec. 8. we see how this problem can be mostly alleviated in practice.**

## 7.2 Brief Comparison against other multivariate Schemes

TTS belongs to the multivariate quadratic finite-field-based family of signature schemes as do two of the five NESSIE ([19]) digital signature scheme candidates, QUARTZ ([4]) and SFLASH ([5]). In any scheme of this type, to verify a message means substitutions into a set of quadratic polynomials much like encrypting a message in any such Public Key Cryptosystem, and hence they all use comparable time and resources. Indeed both QUARTZ and SFLASH performs similarly here.

However, signing a message is really like decrypting a message under an analogous cryptosystem, and here we see substantial performance differences (Table 2). TTS is structured so that signing is also substitutions, 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 Table 2 that TTS shines against (at least) these rival schemes for digital signatures.

| | QUARTZ | SFLASH | TTS-0 | TTS-r |
|---|---|---|---|---|
| Signature (bits) | 128 | 259 | 192 | 224 |
| Public key (kB) | 71 | 15.4 | 6.4 | 8.6 |
| Private key (kB) | 3 | 2.4 | 1.1 | 1.3 |
| Generate keys (sec) | $\approx 4$ | $\approx 1$ | $\approx 10^{-2}$ | $\approx 10^{-2}$ |
| Signing speed (sec) | $\approx 10$ | $2.7 \times 10^{-3}$ | $\approx 10^{-4}$ | $\approx 10^{-4}$ |
| Verifying speed (sec) | $\approx 10^{-3}$ | $8 \times 10^{-4}$ | $\approx 10^{-3}$ | $\approx 10^{-3}$ |

Table 2. Comparison of signature schemes (execution times on a Pentium III/500, non-optimized).

As mentioned in Sec. 2., the underlying algebraic structure is why TTS signs and generates keys faster than SFLASH or QUARTZ. Certainly the use of $\mathrm{GF}(2^7)$ in SFLASH (instead of $\mathrm{GF}(2^8)$) precludes exploiting the existence of intermediate field extensions. Embarassingly, QUARTZ can fail to sign a message (very low probability), and the culprit is again the underlying structure.

It is often said, also, that variable terms in the central non-affine map as TTM/TTS have (as opposed to SFLASH, with a simple fixed central portion) is a plus. SFLASH can thus be subjected to the kind of attack tailored by Steinwandt *et al* ([23]-[25]).

## 8. Key Size Reduction

As can be seen from Sec. 7., TTS keys (while still shorter than SFLASH keys) are longer than RSA keys. This is not significant for modern general-purpose hardware, but can definitely be a problem for embedded applications. Since an embedded cryptosystem application, such as a smart card, principally restricts the size of **private** keys, we will show how to make keys (mostly private keys) significantly smaller by restricting the mappings involved. **The following applies just fine to TTM and most multivariate quadratic finite-field-based schemes.** While complexity of cracking TTS would also decrease, but analysis (Sec. 9.) shows that it is still a "strong cryptosystem" and beyond the reach of crackers in the foreseeable future.

### 8.1 Public Keys

We fix a 4-bit subfield $K' = \mathrm{GF}(2^4)$ that is compatible (has a compatible multiplication table) with our $\mathrm{GF}(2^8)$, and use elements from $K'$ for all the quadratic terms in the Tame Transformation and all matrix elements in the affine mappings. This effectively halves the public key size and can **nearly double the execution speed with suitable programming exercises**.

### 8.2 Private Keys

While public TTS keys are just coefficients for quadratic polynomials, a private TTS key is the sum of its three parts, and there are more tricks available:

1. Using a compatible $K' = \mathrm{GF}(2^4)$ cuts the corresponding private key size.

2. 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 trick lets the programmer avoid costly table lookups and use a variety of SIMD techniques to do several operations in parallel. In some cases it is not even required that the CPU has SIMD instructions!

3. 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 a very much rather smaller $(28+28+28)+(20+20+20) = 144$ bytes in private keys. It also cuts the running time from $O(s^2)$ to $O(s)$, where $s$ is the number of bytes in the signature.

A test implementation of TTS-0 ($20 \rightarrow 24$) used tricks 1 and 3: 4-bit parameters are used for $\phi_2$, for a total 40 bytes (instead of 80). Both $\phi_3$ and $\phi_1$ are written in a $M = L(J - P)R$ form; both $P_1$ and $P_3$ are specified with packed 5-bit entries for each row, cutting 44 bytes down to five eighth that many (28 bytes). $L_1$, $L_3$, $R_1$ and $R_3$ (4-bit entries again!) are 44 bytes more. Constants $\mathbf{c}_1$ and $\mathbf{c}_3$ are 1 byte per entry (44 bytes). **The private key weighs in at 156 bytes[3], shorter than RSA-2048 (and not much longer than RSA-1024).**

## 9. Brief Cryptanalysis

Solving quadratic systems is an NP-complete problem ([8], [12]), so no attack can do more damage than brute force unless there is an inherent flaw in TTS. Given the kinship between TTM and TTS, most attacks against TTM may be surmised to apply to TTS, but there are signatures-specific attacks, too.

### 9.1 Recap of TTM attacks

Some notable attempts against TTM are as follows:

1. Jacques Patarin in [20] and later [21] 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 $y_j = f_j(x_1, \ldots, x_m)$, $j = 1 \cdots n$.) $P$ and $Q$, find affine (linear) mappings $L_1$ and $L_2$ such that $Q = L_1 \circ P \circ$

[3] $\mathbf{c}_3$ can be omitted, going down to 132 bytes, if computing power is not a problem!

$L_2$. (Reduction to) IP is ineffective against TTM because (see [17]) 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. A. Kipnis and A. 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 relieve to some extent the problem of too many extraneous solutions and is used against HFE in [11]. However as is shown in [15] such an attack does not decrease the amount of computations needed for solving many implementations of TTM to a manageable level.

3. In [6] 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 "overdefined 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 [16] that such attempts are futile according to the theory established by Hilbert and Serre, because for XL to have any efficacy the solution set at $\infty$ must be either empty or 0-dimensional.

4. In [4] L. Goubin and N. Courtois asserted that the **MinRank attack** is effective against TTM. Unfortunately the paper was so full of inaccuracies as pointed out in [18], that it is hard to see how the 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.

We conclude that an analog of an attack on TTM (see [14] for details) on TTS will not do much better than a brute force search and all reasonable implementations of TTS has an effective complexity well above $2^{80}$.

## 9.2 Signature-specific attacks

One type of attack depend specifically on the central non-affine mapping having a simple, fixed form, as in the aforementioned IP and the attack on SFLASH by Steinwandt *et al* ([23]-[25]). These kind of attacks are not applicable to TTS. Another case in which an effective attack was tailored to solve "underdefined" systems ([10]) is the the number of variables $n \geq m(m+1)$ ($m$ is the number of equations) for fields of characteristic 2. In [3], the authors generalize this to three new attacks. We evaluate each proposed attack on TTS-0 ($20 \rightarrow 24$), a currently testing variant with $q = 2^8$, $n = 24$ and $m = 20$.

**A** The complexity is $O(q^{m-k}) \approx 2^{144}$ where the constant $k$ is $\min(m/2, \left\lfloor \sqrt{n/2 - \sqrt{n/2}} \right\rfloor)$, equal to 2 in this instance.

**B** The complexity is $K \cdot q^{m-k}$. Here $k \equiv \lfloor \sqrt{2m+2} - 1.5 \rfloor = 4$ and $q^{m-k} = 2^{128}$. $K = \max(C_2, C_3)$ where $C_3$ is the time needed to solve a system of 4 equations (36 multiplications) and $C_2$ is given as $O(k\,(m-k)^2) \approx 2^{10}$, making for a complexity of $2^{138}$.

**C** Not applicable to TTS since it requires $n \geq 2m$.

**We conclude that TTS appears to be reasonably safe, even in its most basic form.**

## 10. A Fast, Flexible New Hash Function: Introducing TTH

[22] gave these excellent criteria for hash functions:

> 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 common 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 $D_1, D_2, \ldots, D_n$. Using the DES algorithm E with secret

key K, the DAC (data authentication code) is computed as follows:

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

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 the 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 $D_1, \ldots, D_n$, then recursively construct the following:

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

   where $C$ is a random $k$-byte vector and $T_\pi$ is a set of generic quadratic polynomials with its coefficients picked from the binary expansion of $\pi$ (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 ([8], [12]).

   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 adjustible. 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. and run the same program.

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.


## 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 second author would like to thank the MIT Mathematics Department's most capable Shirley and Camille as well as Profs. Kleitman and Stanley for their hospitality during his stay in the school year 2001-02. He was also partially sponsored by a National Science Council (of R.O.C.) Grant during this time.

Due to space limitations for the proceedings, some details have been removed from the original manuscript. The original can be found at `http://www.chnds.com.tw` or `http://www.usdsi.com` or consult the *Cryptology ePrint Archive* on International Association for Cryptographic Research website (`http://eprint.iacr.org`).


## References


[1] `http://csrc.nist.gov/encryption/aes` the AES homepage

[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, L. Goubin, W. Meier, and J.-D. Tacier, *Solving Underdefined Systems of Quadratic Equations*, pp. 211–227 in *Public Key Cryptography – PKC 2002*, LNCS V. 2274, Springer-Verlag, 2002.

[4] N. Courtois, L. Goubin, and J. Patarin, *Quartz, 128-bit long digital signatures.* in *Cryptographers' Track RSA Conference 2001*, LNCS V. *2020*, Springer-Verlag, 2001.

[5] N. Courtois, L. Goubin, and J. Patarin, *FLASH, a fast multivariate signature algorithm.* in *Cryptographers' Track RSA Conference 2001*, LNCS V. *2020*, Springer-Verlag, 2001; updated version of [4] and [5] can be found at `http://www.cosic.esat.kuleuven.ac.be/nessie/tweaks.html`

[6] N. Courtois., A. Klimov, J. Patarin and A. Shamir. *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations* pp. 392–407 in EUROCRYPT *2000*, LNCS V. *1807*, Springer-Verlag, 2000.

[7] Joan Daemen and Vincent Rijmen, *The Design of Rijndael, AES - The Advanced Encryption Standard.* Springer-Verlag, 2002.

[8] Michael Garey and David Johnson, *Computers and Intractability, a guide to the theory of NP-completeness.* Freeman, 1979.

[9] L. Goubin and N. Courtois, *Cryptanalysis of the TTM Cryptosystem.* pp. 44–57 in ASIACRYPT *2000*, LNCS V. *1976*, Springer-Verlag, 2000.

[10] A. Kipnis, J. Patarin, and L. Goubin, *Unbalanced Oil and Vinegar Signature Schemes*, pp. 206-222 in EUROCRYPT*'99*, LNCS V. *1592*, Springer-Verlag, 1999.

[11] A. Kipnis and A. Shamir, *Crypanalysis of the HFE public key cryptosystem*, pp. 19–30 in CRYPTO *1999*, LNCS V. *1666*, Springer-Verlag, 1999.

[12] 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.

[13] B. Lucier, *Cryptography, Finite Fields, and AltiVec*, `http://www.altivec.org/articles/` has preprint available.

[14] T. Moh, *A Public Key System with Signature and Master Key Functions.* Communications in Algebra, 27 (1999) 2207–2222.

[15] T. Moh, *Relinearization and TTM*, Preprint 1999.

[16] T. Moh, *On The Method of XL and Its Inefficiency Against TTM* in Cryptology ePrint Archive (2001/47).

[17] T. Moh, *A Cryptanalysis of TTM* in *Multivariate Cryptography*, International Press, 2003.

[18] 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.tw`.

[19] `http://www.cosic.esat.kuleuven.ac.be/nessie/`, the NESSIE (New European Schemes for Signatures, Integrity, and Encryption) selection project homepage.

[20] Jacques Patarin, *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms*, pp. 33–48 in EUROCRYPT *1996*, LNCS V. *1070*, Springer-Verlag, 1996.

[21] J. Patarin, L. Goubin and N. Courtois, *Improved Algorithms for Isomorphisms of Polynomials*, pp. 184–200 in EUROCRYPT *1998*, LNCS V. *1403*, Springer-Verlag, 1998.

[22] W. Stallings, *Crytography and Network Security: Principles and Practice*, 2nd ed. Prentice Hall, 1998.

[23] R. Steinwandt, W. Geiselmann, and Th. Beth, *Revealing the Affine Parts of* SFLASH$^{v1}$ SFLASH$^{v2}$, *and* FLASH, to appear in *VII Reunion Espanola sobre Criptologia y Seguridad de la Informacion, VII RECSI Proceedings*.

[24] R. Steinwandt, W. Geiselmann, and Th. Beth, *Attacking the Affine Parts of* SFLASH, pp. 355-359, in *Cryptography and Coding, 8th IMA International Conference Proceedings*, B. Honary, ed., LNCS V. 2260, Springer-Verlag, 2001.

[25] R. Steinwandt, W. Geiselmann, and Th. Beth, *A Theoretical DPA-Based Cryptanalysis of the NESSIE Candidates* FLASH *and* SFLASH, pp. 280-293 in *Information Security, 4th International Conference, ISC 2001 Proceedings*, G. I. Davida, Y. Frankel, eds., LNCS V. 2200, Springer-Verlag, 2001. The last two papers are also presented at the 2nd NESSIE workshop.