Analysis of QUAD

Bo-Yin Yang[1], Owen Chia-Hsin Chen[2],
Daniel J. Bernstein[3], and Jiun-Ming Chen[4]

[1] Academia Sinica and Taiwan Information Security Center, by@moscito.org
[2] National Taiwan University, owenhsin@gmail.com
[3] University of Illinois at Chicago, djb@cr.yp.to
[4] Nat'l Taiwan University and Nat'l Cheng Kung University, jmchen@ntu.edu.tw

Abstract. In a Eurocrypt 2006 article entitled "QUAD: A Practical Stream Cipher with Provable Security," Berbain, Gilbert, and Patarin introduced QUAD, a parametrized family of stream ciphers. The article stated that "the security of the novel stream cipher is provably reducible to the intractability of the MQ problem"; this reduction deduces the infeasibility of attacks on QUAD from the hypothesized infeasibility (with an extra looseness factor) of attacks on the well-known hard problem of solving systems of multivariate quadratic equations over finite fields. The QUAD talk at Eurocrypt 2006 reported speeds for QUAD instances with 160-bit state and output block over the fields GF(2), GF(16), and GF(256).

This paper discusses both theoretical and practical aspects of attacking QUAD and of attacking the underlying hard problem. For example, this paper shows how to use XL-Wiedemann to break the GF(256) instance QUAD$(256, 20, 20)$ in approximately 2^{66} Opteron cycles, and to break the underlying hard problem in approximately 2^{45} cycles. For each of the QUAD parameters presented at Eurocrypt 2006, this analysis shows the implications and limitations of the security proofs, pointing out which QUAD instances are not secure, and which ones will never be proven secure. Empirical data backs up the theoretical conclusions; in particular, the 2^{45}-cycle attack was carried out successfully.

1 Introduction

1.1 Questions

Berbain, Gilbert, and Patarin introduced QUAD at Eurocrypt 2006. Their article [7] is titled "A Practical Stream Cipher," but QUAD is not actually a single stream

* The authors would all like to thank the TWISC project (Taiwan Information Security Center, NSC95-2218-E-001-001) for sponsoring a series of lectures on cryptology at its Nat'l Taiwan U. of Sci. and Tech. location (NSC95-2218-E-011-015) in 2006, the discussions following which led to this work.

* BY would like to thank the Taiwan's National Science Council for support under project 95-2115-M-001-021 and indirectly via TWISC.

* JMC was partially supported by TWISC at NCKU (NSC94-3114-P-006-001-Y).

* Date of this document: 2007.04.18.

cipher; it is a parametrized family of stream ciphers. The most important parameters are the `QUAD` field size, the number of `QUAD` variables, and the number of `QUAD` outputs per round. We will write $\texttt{QUAD}(q, n, r)$ to mean any instance of `QUAD` that has field size q, uses n variables, and produces r outputs per round.

The speed and security of `QUAD` are (obviously) functions of the `QUAD` parameters. The Performance section [7, Section 6] of the `QUAD` paper reported a speed of 2915 Pentium-IV cycles/byte for the "recommended version of `QUAD`," namely $\texttt{QUAD}(2, 160, 160)$. The section closed with the following intriguing comment: "Though `QUAD` is significantly slower than AES, which runs at 25 cycles/byte, it is much more efficient than other provably secure pseudo random generator. Moreover, implementations of `QUAD` with quadratic system over larger fields (e.g. GF(16) or GF(256) are much faster and even reach 106 cycles/byte."

Further performance details were revealed in the `QUAD` talk at Eurocrypt 2006. In that talk, the `QUAD` authors reported the following implementation results:

- $\texttt{QUAD}(2, 160, 160)$: 2930 cycles/byte for 32-bit architecture, 2081 for 64-bit;
- $\texttt{QUAD}(16, 40, 40)$: 990 cycles/byte for 32-bit architecture, 745 for 64-bit;
- $\texttt{QUAD}(256, 20, 20)$: 530 cycles/byte for 32-bit architecture, 417 (confirmed in [5], correcting the erroneous "106" in [7]) for 64-bit.

We wrote privately to the `QUAD` authors to ask about parameters. The authors confirmed in response that the Eurocrypt 2006 speed reports were for "160 bits of internal state, and 160 bits produced at each round," i.e., $\texttt{QUAD}(q, n, n)$ with $q^n = 2^{160}$; and that the talk reported the above speeds for $q = 2$, 16, and 256. Speed was never reported for the "proven secure" $\texttt{QUAD}(2, 256, 256)$ or $\texttt{QUAD}(2, 350, 350)$.

Which of these `QUAD` instances are actually secure? What is the security level under the best known attack? What does the "provable security" of `QUAD` mean for these parameter choices? What about other parameter choices?

1.2 Conclusions

This paper discusses both theoretical and practical aspects of solving the nonlinear multivariate systems associated with the security of `QUAD`. Our analysis produces the following conclusions regarding the security of `QUAD`:

- Section 4: For a surprisingly wide range of parameters (which we call "broken" parameters), a feasible computation distinguishes the `QUAD` output stream from uniform; in fact, an attacker can compute `QUAD`'s secret internal state from a few output blocks. For example, $\texttt{QUAD}(256, 20, 20)$, one of the three `QUAD` instances for which timings were reported in the `QUAD` talks at Eurocrypt 2006 and SAC 2006 [5], is breakable in no more than 2^{66} cycles.
- Section 5: For an even wider range of parameters (which we call "unprovable"), a feasible attack breaks the "hard problem" underlying `QUAD`. For example, the "hard problems" underlying $\texttt{QUAD}(16, 40, 40)$ and $\texttt{QUAD}(256, 20, 20)$ are breakable in approximately 2^{71} and 2^{45} cycles respectively. An "unprovable" instance can *never* be provably secure. *This does not mean that the `QUAD` instance is breakable!*

- Section 6: For an extremely wide range of parameters (which we call "unproven"), including QUAD$(256, 20, 20)$, QUAD$(16, 40, 40)$, QUAD$(2, 160, 160)$, and QUAD$(2, 256, 256)$, a "loosely feasible" attack breaks the "hard problem" underlying QUAD. "Loosely feasible" means that the attack time is smaller than $2^{80}L$ where L is the looseness factor in QUAD's "provable security" theorem. *This does not mean that the proof is in error, and it does not mean that the QUAD instance is breakable!*

Our algorithm analysis—in particular, our analysis of XL-Wiedemann—may also be useful for readers interested in other applications of solving systems of multivariate equations.

We do not claim that *all* QUAD parameters are broken, or unprovable, or unproven. All of our attacks against QUAD(q, n, n) have cost growing exponentially with n when q is fixed. The problem for QUAD is that—especially for large q—the base of the exponential is surprisingly small, so n must be surprisingly large to achieve proven security against these attacks.

We see ample justification for future investigation of QUAD. The QUAD security argument is that a feasible attack against QUAD(q, n, n) for moderately large n would imply an advance in attacks against the $\mathcal{MQ}$ problem, the problem of solving systems of multivariate quadratic equations over finite fields:

Problem $\mathcal{MQ}$: Solve the system $P_1 = P_2 = \cdots = P_m = 0$, where each P_i is a quadratic polynomial in $\mathbf{x} = (x_1, \ldots, x_n)$. All coefficients and variables are in the field $K = \mathrm{GF}(q)$.

This is a well-known difficult problem [20] and the basis for multivariate-quadratic public-key cryptosystems [16, 27]. However, we are concerned by the choice of QUAD$(2, 160, 160)$, QUAD$(16, 40, 40)$, and QUAD$(256, 20, 20)$ as the subjects of speed reports in the QUAD talk at Eurocrypt 2006. QUAD$(256, 20, 20)$ is now broken; no extension of the security arguments in [7] will ever prove QUAD$(16, 40, 40)$ secure; and QUAD$(2, 160, 160)$, while apparently unbroken, is currently rather far from being provably secure. There is no single QUAD stream cipher that simultaneously provides the advertised levels of speed and provable security. We recommend that future QUAD evaluations stop describing QUAD as "a stream cipher" and start carefully distinguishing between different choices of the QUAD parameters.

1.3 Previous work

QUAD is a new proposal, but we are certainly not the first to observe difficulties in the parameter choices for "provably secure" cryptosystems.

Consider, for example, the famous BBS stream generator. Blum, Blum, and Shub [8] proved that an attack against this generator can be converted into an integer-factorization algorithm with a polynomially bounded loss of efficiency and effectiveness. Koblitz and Menezes [22, Section 6.1], after comparing the latest refinements in the BBS security theorem, the speeds of existing factorization algorithms, and the BBS parameter choices in the literature, concluded that common BBS parameter choices destroy the "provable security" of BBS.

The speed comparison between 1024-bit BBS and QUAD$(2, 160, 160)$ in [7, Section 6] issues the same warning regarding 1024-bit BBS ("far from the number of bits of the internal state required for proven security"), but does not highlight the comparable lack of proof for "recommended" QUAD$(2, 160, 160)$. See [21] and [22] for more discussions of the limits of "provable security."

Of course, a cryptosystem can be unbroken, and perhaps acceptable to users, without having a security proof. We are not aware of any feasible attacks on any proposed BBS parameter choices. The situation for QUAD is qualitatively different: The QUAD authors reported speeds for QUAD$(256, 20, 20)$. This cipher is not merely unprovable but has now actually been broken.

The QUAD paper contains some analysis of parameter choices: [7, page 121] summarizes Bardet's estimates [2] of equation-solving time for $q = 2$, and concludes that there is no "contradiction"—i.e., the QUAD security proof does not guarantee 80-bit security for 2^{40} output bits against known attacks—for QUAD$(2, n, n)$ for $n < 350$. We analyze QUAD in much more depth. We go beyond $q = 2$, showing that QUAD$(16, 40, 40)$ is unprovable; we consider not just the cost of solving the underlying "hard problem" but also the extra cost of attacking QUAD, showing that QUAD$(256, 20, 20)$ is not merely unprovable but also broken; and we cover both theoretical and practical attack complexity, for example discussing the implications of parallelizing communication costs.

1.4 Future work

We suggest using the same classification of levels of danger for parameter choices in other "provably secure" systems—systems having security theorems that deduce the infeasibility of attacks on the cryptosystem, assuming the loose infeasibility of attacks on the underlying "hard problem":

- Unproven parameter choices (e.g., 2048-bit BBS or QUAD$(2, 160, 160)$ as "recommended"): Known attacks on the underlying "hard problem" are loosely feasible. They might not be feasible, but the gap is smaller than the looseness of the security proof. It is unjustified and somewhat misleading to label these parameters as "provably secure."
- Unprovable parameter choices (e.g., 512-bit BBS or QUAD$(16, 40, 40)$): Known attacks on the underlying "hard problem" are feasible. These parameters would not be "provably secure" even if the proof were tightened.
- Broken parameter choices (e.g., QUAD$(256, 20, 20)$): Feasible attacks are known on the cryptosystem per se. Users must avoid these parameters.

It would be interesting to see a careful comparison of speeds for "provably secure" systems with parameters that avoid all of these dangers. This is a quite different task from comparing systems such as 1024-bit BBS and QUAD$(2, 160, 160)$, both of which are unbroken but have no security proof; the motivation for "provably secure" systems does not apply to unproven parameter choices.

Our security analysis is essentially independent of the choice of polynomials in QUAD. [7, page 113, second paragraph] suggests, but neither quantifies nor justifies, a few "extra precautions" regarding "weak" choices of polynomials. Are some polynomial choices weaker than others? For example, one can save stream-generation time by reducing the density of the quadratic polynomials in QUAD from 1/2 to some small ε; what effect does this have on security? How much time can we save in our attacks as ε decreases? Exactly how small would ε have to be before the Raddum-Semaev attack [26] becomes a problem? There are many obvious directions for future security analysis.

2 The QUAD Family of Stream Ciphers

2.1 Definition of QUAD

To specify a particular QUAD stream cipher one must specify a prime power q; positive integers n and r; an "output filter" $\mathbf{P} : \mathrm{GF}(q)^n \to \mathrm{GF}(q)^r$ consisting of r quadratic polynomials $P_1, P_2, \ldots, P_r$ in n variables; and an "update function" $\mathbf{Q} : \mathrm{GF}(q)^n \to \mathrm{GF}(q)^n$ consisting of n quadratic polynomials $Q_1, Q_2, \ldots, Q_n$ in n variables. These quantities $q, n, r, \mathbf{P}, \mathbf{Q}$ are not meant to be secret; they can be published and standardized.

The QUAD cipher expands a secret initial state $\mathbf{x}_0 \in \mathrm{GF}(q)^n$ into a sequence of secret states $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots \in \mathrm{GF}(q)^n$ and a sequence of output vectors $\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots \in \mathrm{GF}(q)^r$ as follows:

$$\begin{tikzcd}
\mathbf{x}_0 \arrow[r] \arrow[d] & \mathbf{x}_1 = \mathbf{Q}(\mathbf{x}_0) \arrow[r] \arrow[d] & \mathbf{x}_2 = \mathbf{Q}(\mathbf{x}_1) \arrow[r] \arrow[d] & \mathbf{x}_3 = \mathbf{Q}(\mathbf{x}_2) \arrow[r] \arrow[d] & \cdots \\
\mathbf{y}_0 = \mathbf{P}(\mathbf{x}_0) & \mathbf{y}_1 = \mathbf{P}(\mathbf{x}_1) & \mathbf{y}_2 = \mathbf{P}(\mathbf{x}_2) & \mathbf{y}_3 = \mathbf{P}(\mathbf{x}_3) & \cdots
\end{tikzcd}$$

Typically q is a power of 2, allowing each output vector $\mathbf{y}_i \in \mathrm{GF}(q)^r$ to encrypt the next $r \lg q$ bits of plaintext in a straightforward way.

2.2 Parameter restrictions

Can users expect QUAD to be secure no matter how the parameters are chosen? Certainly not. For example, QUAD$(2, 20, 20, \mathbf{P}, \mathbf{Q})$ has only 20 bits in its initial state $\mathbf{x}_0$, so the initial state can be discovered from some known plaintext by a brute-force search. As another example, QUAD$(2, 512, 512, 0, \mathbf{Q})$ is the "all-zero cipher," a silly stream cipher that always outputs 0. There are other, less obvious, attacks; users considering QUAD need to understand which parameters are broken or potentially so.

The QUAD paper [7, page 114] requires that the public polynomials $P_1, \ldots, P_r$ and $Q_1, \ldots, Q_n$ be "chosen randomly." We see conflicting statements regarding the distribution of these random variables: [7, page 112] defines "chosen randomly" as a uniform random choice (coefficients "uniformly and independently drawn"), but [7, page 113] suggests checking for and discarding some choices. Either way, one can reasonably conjecture that "bad" choices of $\mathbf{P}$ and $\mathbf{Q}$ have a negligible chance of occurring. There is no proof of this conjecture, and as mentioned in the introduction we would like to know exactly how $\mathbf{P}$ and $\mathbf{Q}$ affect security, but that is not the focus of this paper; we consider attacks that work for most choices of $\mathbf{P}$ and $\mathbf{Q}$.

2.3 Example: QUAD(256, 20, 20)

Choose $q = 256$, $n = 20$, and $r = 20$. Also choose 40 quadratic polynomials $P_1, \ldots, P_{20}, Q_1, \ldots, Q_{20}$ in 20 variables. These choices specify a particular QUAD stream cipher. The cipher starts with a secret 20-byte state $\mathbf{x}_0 \in \mathrm{GF}(256)^{20}$; computes another secret 20-byte state $\mathbf{x}_1 = (Q_1(\mathbf{x}_0), \ldots, Q_{20}(\mathbf{x}_0))$; computes another secret 20-byte state $\mathbf{x}_2 = (Q_1(\mathbf{x}_1), \ldots, Q_{20}(\mathbf{x}_1))$; and so on. The cipher outputs $\mathbf{y}_0 = (P_1(\mathbf{x}_0), \ldots, P_{20}(\mathbf{x}_0))$; $\mathbf{y}_1 = (P_1(\mathbf{x}_1), \ldots, P_{20}(\mathbf{x}_1))$; $\mathbf{y}_2 = (P_1(\mathbf{x}_2), \ldots, P_{20}(\mathbf{x}_2))$; and so on.

We specifically warn against using QUAD$(256, 20, 20)$. Given the public polynomials $P_1, \ldots, P_{20}, Q_1, \ldots, Q_{20}$ and the first 40 bytes $\mathbf{y}_0, \mathbf{y}_1$ of the output stream, our attacks compute the secret 20-byte initial state $\mathbf{x}_0$ in approximately 2^{66} cycles. See Section 4.2 for details.

2.4 Nonces

Old-fashioned stream ciphers, such as RC4, compute an output stream starting from a secret key. Modern stream ciphers, such as the eSTREAM submissions, compute an output stream starting from a secret key *and a nonce*, allowing the same secret key to be used for many separate output streams.

We have presented QUAD as an old-fashioned stream cipher. The QUAD paper actually presents a modern stream cipher. The modern stream cipher begins with an "initialization," converting a secret key and a nonce into a secret initial state $\mathbf{x}_0$. The modern stream cipher then generates output in exactly the way we have described. Details of the initialization are not relevant to this paper; our attacks recover $\mathbf{x}_0$ no matter how $\mathbf{x}_0$ was generated, breaking both the old-fashioned stream cipher and the modern stream cipher.

Stream ciphers can have "refresh" rules to disrupt the state after some given number of clocks. We do not concern ourselves with this either: initialization and refresh are both considered to be perfectly secure hereafter. For reference, [6] proves one particular setup procedure secure *with an extra loss of efficiency.*

3 How to Solve Multivariate Systems

Modern methods for system all descend spiritually from Buchberger's algorithm of finding Gröbner Bases [9], which is still widespread in symbolic mathematics packages both commercial and free (e.g., Maple and Singular). We present some state-of-the-art improvements below, not nearly as well known in general.

3.1 History of Lazard-Faugère solvers: $\mathbf{F_4}$, $\mathbf{F_5}$, XL, XL2, FXL

Macaulay generalized Sylvester's matrix to multivariate polynomials [24]. The idea is to construct a matrix whose lines contain the multiples of the polynomials in the original system, the columns representing a basis of monomials up to a given degree. It was observed by D. Lazard [23] that for a large enough degree, ordering the columns according to a monomial ordering and performing row reduction without column pivoting on the matrix is equivalent to Buchberger's algorithm. In this correspondence, reductions to 0 correspond to lines that are linearly dependent upon the previous ones and the leading term of a polynomial is given by the leftmost nonzero entry in the corresponding line.

Lazard's idea was rediscovered in 1999 by Courtois, Klimov, Patarin, and Shamir [12] as **XL**. Courtois *et al* proposed several adjuncts [11, 13, 14] to XL. One tweak called XL2 merits a mention as an easy to understand precursor to $\mathbf{F_4}$. Another of these proved to be a real improvement to $\mathbf{F_4}/\mathbf{F_5}$ as well as XL. This is FXL, where F means "fixing" (guessing at) variables.

Some time prior to this, J. -C. Faugère had proposed a much improved Gröbner bases algorithm called $\mathbf{F_4}$ [17]. A later version, $\mathbf{F_5}$ [18], made headlines [19] when it was used to solve HFE Challenge 1 in 2002. Commercially, $\mathbf{F_4}$ is only implemented in the computer algebra system MAGMA [10].

3.2 Algorithm XL (eXtended Linearization) at degree D

For the rest of this paper, we will denote the monomial $x_1^{b_1}x_2^{b_2}\cdots x_n^{b_n}$ by $\mathbf{x^b}$, and its total degree $|\mathbf{b}| = b_1 + \cdots + b_n$. The set of degree-$D$-or-lower monomials is denoted $\mathcal{T} = \mathcal{T}^{(D)} = \{\mathbf{x^b} : |\mathbf{b}| \le D\}$. $|\mathcal{T}|$ is the number of degree $\le D$ monomials and denoted $T^{(D)} = T$.

Start by multiplying each equation p_i, $i = 1\cdots m$ by all monomials $\mathbf{x^b} \in \mathcal{T}^{(D-2)}$. Reduce as a linear system of the equations $\mathcal{R}^{(D)} = \{\mathbf{x^b}p_j(\mathbf{x}) = 0 : 1 \le j \le m, |\mathbf{b}| \le D-2\}$, with the monomials $\mathbf{x^b} \in \mathcal{T}^{(D)}$ as independent variables. Repeat with higher D until we have a solution, a contradiction, or reduce the system to a univariate equation in some variable. The number of equations and independent equations are denoted $R^{(D)} = R = |\mathcal{R}|$ and $I^{(D)} = I = \dim(\mathrm{span}\mathcal{R})$.

If we accept solutions in arbitrary extensions of $K = \mathrm{GF}(q)$, then $T = \binom{n+D}{D}$ regardless of q. However, most crypto applications require solutions in $\mathrm{GF}(q)$ only. The above expression for T then only holds for large q, since we may identify x_i^q with x_i and cut substantially the number of monomials we need to manage. This "Reduced XL" (cf. C. Diem [15]) can lead to extreme savings compared to "Original XL," e.g., if $q = 2$, then $T = \sum_{j=0}^{D}\binom{n}{j}$.

Proposition 3.3 ([3, 30]) *The number of monomials is* $T = [t^D]\ \dfrac{(1-t^q)^n}{(1-t)^{n+1}}$ *which reduces to* $\binom{n+D}{D}$ *when* q *is large. We can then find* $R = R^{(D)} = mT^{(D-2)}$.

We note that the XL of [12, 13] terminates more or less reliably when $T - I \le \min(D, q-1)$, but sparse matrix computation is only possible when $T - I \le 1$

[29]. Further, Lazard-Faugère methods work for equations of any degree [4, 30]. If $\deg(p_i) = d$, we will only multiply the equation p_i with monomials up to degree $D - d$ in generating $\mathcal{R}^{(D)}$. The principal result is:

Proposition 3.4 ([30, Theorem 7]) *If the equations p_i, with $\deg p_i := d_i$, and **the relations $\mathcal{R}^{(D)}$ has no other dependencies than the obvious ones generated by** $p_i p_j = p_j p_i$ **and** $p_i^q = p_i$, then*

$$T - I = [t^D]\ G(t) = [t^D]\ \frac{(1-t^q)^n}{(1-t)^{n+1}} \prod_{j=1}^{m} \left(\frac{1-t^{d_j}}{1-t^{q\,d_j}} \right). \qquad (1)$$

After a certain degree D_{XL}, called the degree of regularity for XL, such that $D_{XL} := \min\{D : [t^D]\ G(t) \le 0\}$ is the smallest D such that Eq. 1 cannot hold if the system has a solution, because the right hand side of Eq. 1 goes nonpositive. If the boldfaced condition above holds for as long as possible (which means for degrees up to D_{XL}), we say that the system is **K-semi-regular** or **q-semi-regular** (cf. [3, 30]). Diem proves [15] for char 0 fields, and conjectures for all K that (i) a generic system (no algebraic relationship betweem the coefficients) is K-semi-regular and (b) if $(p_i)_{i=1\cdots m}$ are *not* K-semi-regular, I can only decrease from the Eq. 1 prediction. Most experts [15] believe the conjecture that a *random* system behaves like a generic system with probability close to 1.

Corollary 3.5 $T - I = [t^D] \left((1-t)^{-n-1} \prod_{j=1}^{m} \left(1 - t^{d_i}\right) \right)$ *for generic equations if $D \le \min(q, D_{XL}^{\infty})$, where D_{XL}^{∞} is the degree of the lowest term with a non-positive coefficient in* $G(t) = \left((1-t)^{-n-1} \prod_{j=1}^{m} \left(1 - t^{d_i}\right) \right)$.

We would note that (F)XL is a solver and not a true Gröbner basis method as $\mathbf{F_4}$ and $\mathbf{F_5}$ is. However, the analysis much parallels that of $\mathbf{F_4}$-$\mathbf{F_5}$ by Dr. Faugère et al, hence the categorical name "Lazard-Faugère" solvers.

3.6 XL2, $\mathbf{F_4}$ and $\mathbf{F_5}$

XL2 [13] is a tweak of XL as follows: Tag each equation with its maximal degree. Run an elimination on the system with monomials in degree-reverse-lex. In the remaining (row echelon form) system, multiply by each variable $x_1, x_2 \cdots$ *all remaining equations with the maximum tagged degree* and eliminate again. When we cannot eliminate all remaining monomials of the maximum degree, increment the operating degree and reallocate more memory.

XL+XL2 can be considered a primitive or inferior matrix form of $\mathbf{F_4}$ or $\mathbf{F_5}$ [1]. $\mathbf{F_4}$ inserts elimination between expansion stages, which compresses the number of rows that needs to be handled. $\mathbf{F_5}$ is a further refinement of $\mathbf{F_4}$. The set of equations is actually generated one by one (or the matrix row by row). In the process, an algebraic criterion is used to determine, ahead of an elimination process, whether a row will be reduced to zero or not and only the meaningful rows are retained. A complication resulting from the tagging is that the elimination must be done in a strictly ordered way. This corresponds in the matrix form to no row exchanges in a Gaussian. There are two separate degrees in $\mathbf{F_4}/\mathbf{F_5}$, an apparent operating degree D_{F4} and a higher intrinsic degree equal to that of the equivalent XL system. For the full power of $\mathbf{F_4}$ or $\mathbf{F_5}$, auxillary algorithms such as FGLM are needed. See [17, 18] for complete details.

Proposition 3.7 ([3]) *If the eqs. p_i are q-semi-regular, at the operating degree*

$$D_{reg} := \min \left\{ D : [t^D] \, \frac{(1-t^q)^n}{(1-t)^n} \prod_{i=1}^{m} \left(\frac{1-t^{d_i}}{1-t^{q d_i}} \right) < 0 \right\}$$

both $\mathbf{F_4}$-$\mathbf{F_5}$ will terminate. Note that by specializing to a large field, we find

$$D_{reg}^{\infty} := \min \left\{ D : [t^D] \, (1-t)^{-n} \prod_{i=1}^{m} \left(1 - t^{d_i}\right) < 0 \right\}. \tag{2}$$

If we compare this formula with Cor. 3.5, we see that the only difference is a substitution of n for $n+1$. In other words, we are effectively running with one fewer variable in the large field case. This explains why $\mathbf{F_4}$-$\mathbf{F_5}$ can be much faster than XL. However, the savings is smaller over small fields like GF(2), and even for large fields, removing one variable may not be enough of a savings, because the systems that we aim to solve will spawn millions of monomials (variables). Eliminating in the usual way means that we will run out of memory before time.

According to the description we received from the MAGMA project and Dr. Faugère, even though memory management is very critical, elimination is still relatively straightforward in current implementations of $\mathbf{F_4}$-$\mathbf{F_5}$, and in the process we see reasonably dense matrices, not extremely sparse ones. All said, $\mathbf{F_4}$-$\mathbf{F_5}$ are still the most sophisticated general system-solving algorithms today.

3.8 Practicalities: XL with Wiedemann, and Ramifications

Table 1 lists our tests in solving generic equations with $\mathbf{F_4}$ over GF(256). That we only have 2GB main memory is not critical to our inability to solve equations in the realm of 20 byte-sized variable in as many equations, because we also ran into a SIGMEM (out of memory) error with a 15-variable, 15-equation system on a 16 GB RAM system.

$m-n$	D_{XL}	D_{reg}	$n=9$	$n=10$	$n=11$	$n=12$	$n=13$
0	2^m	m	6.090	46.770	350.530	3322.630	sigmem
1	m	$\lceil \frac{m+1}{2} \rceil$	1.240	8.970	53.730	413.780	2538.870
2	$\lceil \frac{m+1}{2} \rceil$	$\lceil \frac{m+2-\sqrt{m+2}}{2} \rceil$	0.320	2.230	12.450	88.180	436.600

Table 1. System-solving time (sec): MAGMA 2.12, 2GB RAM, Athlon64x2 2.2GHz

Conclusion: We will run out of memory using any Lazard-Faugère solver, including $\mathbf{F_4}$-$\mathbf{F_5}$, if our equation-solving is not tailored to sparse matrices. *Many authors used $T^{2.8}$ (or $T^{2.376}$!) as the cost function. This is why we don't.*

Example 3.1. If we try to attack `QUAD` directly using $\mathbf{F_4}$-$\mathbf{F_5}$, we will be solving 20 variables from 20 quadratic equations and 20 quartic equations over (say GF(256)). We will run into 6906900 monomials at degree 9 with (eventually) a fairly dense matrix. This is not feasible.

We must take advantage of the sparsity of Macaulay matrix. Solving the same system via XL with a Krylov subspace method means that we will run into 30045105 monomials at degree 10, but with a sparse matrix of only 200 or so entries per row with 5 bytes per entry. The whole thing can fit in a single machine with 32GB or (much easier) distributed among a cluster.

Three well-known methods (all utilizing the existence of Krylov subspaces) adapt well to sparse matrices: Conjugate Gradient, Lanczos, and Wiedemann. There are two reasons to prefer Wiedemann. While Lanczos and CG usually takes about N (as opposed to $3N$) multiplications by the matrix A, they are restricted to symmetric matrices. Furthermore, Wiedemann has no worries about the self-orthogonality issue and is easier to program.

Algorithm: XL (with homogenous Wiedemann)

1. Create the extended Macaulay matrix of the system to a certain degree D_{XL}.
2. Randomly delete some rows then add some columns to form a square system, $A\mathbf{x} = 0$ where $\dim A = \beta T + (1-\beta)R$. Usually we can succeed with $\beta = 1$.
3. Apply the homogeneous version of Wiedemann's method to solve for $\mathbf{x}$:
   (a) Set $k = 0$ and $g_0(z) = 1$, and take a random $\mathbf{b}$.
   (b) Choose a random $\mathbf{u}_{k+1}$ [usually the $(k+1)$-st unit vector].
   (c) Find the sequence $\mathbf{u}_{k+1}A^i\mathbf{b}$ starting from $i = 0$ and going up to $2N-1$.
   (d) Apply g_k as a difference operator to this sequence, and run the Berlekamp-Massey over GF(q) on the result to find the minimal polynomial f_{k+1}.
   (e) Set $g_{k+1} := f_{k+1}g_k$ and $k := k+1$. If $\deg(g_k) < N$ and $k < n$, go to (b).
4. Compute the solution $\mathbf{x}$ using the minpoly $f(z) = g_k(z) = c_m z^m + c_{m-1}z^{m-1} + \cdots + c_\ell z^\ell$: Take another random $\mathbf{b}$. Start from $\mathbf{x} = (c_m A^{m-\ell} + c_{m-1}A^{m-\ell-1} + \cdots + c_\ell 1)\mathbf{b}$, continuing to multiply by A until we find a solution to $Ax = 0$.
5. In the event that the random choice in Step 2 went awry and we dropped an essential equation, or if the system had more than one solution to begin with, the nullity ℓ will be more than 1, and we will have to repeat the check below at every point of a linear subspace (q points).
6. Obtain the solution from the last few elements of $\mathbf{x}$ and check its correctness.

Proposition 3.9 (cf. [4, 30]) *The expected running time of XL is roughly $C_{XL} \sim 3\tau T\mathfrak{m}$ where $\tau = kT$ is the total (and k the average) number of terms in an equation, one multiplication $\mathfrak{m} \approx (c_0 + c_1 \lg T)$ cycles, and c_0, c_1 depend on the architecture. E.g. on x86, when $q = 256$, $T < 2^{24}$, each multiplication cost about 14 cycles on a P4 (3 consecutively dependent loads from L1 cache plus change). On a K8 or P-M/Core, it takes 11 cycles on the same serial code, but in x86-64 mode, some loop-unrolling can get it down to about 8.*

4 Broken Parameters for QUAD

4.1 Overview: Using Algebraic Attacks Against QUAD

The QUAD paper [7, Section 5] contains the following statement regarding algebraic attacks: "QUAD was designed to resist algebraic attack techniques. As a matter of fact, the key and IV loading and keystream generation mechanisms of QUAD are based upon the iteration of quadratic systems whose associated equations are conjectured to be computationally impossible to solve."

Can we try to solve $\mathbf{P}(\mathbf{x}_0) = \mathbf{y}_0$ directly? Yes, but even for $q = 256$, $r = 20$, that takes about 2^{80} cycles [29]. We propose instead to solve the equations $\mathbf{P}(\mathbf{x}_0) = \mathbf{y}_0$ and $\mathbf{P}(\mathbf{Q}(\mathbf{x}_0)) = \mathbf{y}_1$ for the initial state $\mathbf{x}_0$ using (F)XL-Wiedemann.

The attacker is given the quadratic polynomials $\mathbf{P}$ and $\mathbf{Q}$, which presumably are public, and the two initial stream-cipher outputs $\mathbf{y}_0, \mathbf{y}_1$, for example from a successful guess regarding the initial bytes of plaintext. The unknown state $\mathbf{x}_0$ consists of n variables from GF(q). The equation $\mathbf{P}(\mathbf{x}_0) = \mathbf{y}_0$ consists of r quadratic equations in those n variables. The equation $\mathbf{P}(\mathbf{Q}(\mathbf{x}_0)) = \mathbf{y}_1$ consists of r quartic equations in those n variables. The attack solves the combined quadratic-quartic system to find $\mathbf{x}_0$. At this point the attacker can compute the subsequent stream-cipher output $\mathbf{y}_2, \mathbf{y}_3, \ldots$.

Note: we can also frame the attack as *given any consecutive stream-cipher output blocks* $\mathbf{y}_i$, $\mathbf{y}_{i+1}$*, compute the state* $\mathbf{x}_i$ *and all subsequent state and output blocks.* This means that known-plaintext attacks will be particularly effective.

The cost of this attack is, aside from negligible setup costs, the cost of solving r quadratic equations and r quartic equations in n variables over GF(q), which in turn is dominated by solving a large sparse matrix equation. In the rest of this section we consider the cost of (F)XL-Wiedemann for various choices of (q, n, r). In particular, we show that the attack is feasible against QUAD$(256, 20, 20)$.

4.2 Example: Breaking QUAD$(256, 20, 20)$

Proposition 4.3 *For* $q > 10$*, a* q*-semiregular system of 20 quadratics and 20 quartics can be solved over* GF(q) *in no more than* 2^{63} GF(q)*-multiplications (for* $q = 256$*, about* 2^{66} *cycles).*

Proof. The minimum degree to find a nonpositive coefficient in $(1-t)^{-21}(1-t^2)^{20}(1-t^4)^{20}$ is $D_{XL} = 10$, we have $T = \binom{30}{10} = 30045015$, the number of initial equations is $R = 20 \times \binom{28}{8} + 20 \times \binom{26}{6} = 66766700$, and the total number of terms in those equations is $\tau = 20\binom{28}{8}\binom{22}{2} + 20\binom{26}{6}\binom{24}{4} = 63287924700$. Hence the number of multiplications needed is bounded by $3T\tau \lesssim 2^{63}$, or about 2^{66} cycles. **If we can cut down to a $T \times T$ system (which usually succeeds), then it takes $3T(\tau T/R) \sim 2^{60}$ multiplications, or about 2^{63} cycles.**

In contrast, QUAD$(16, 40, 40)$ *is not directly breakable by this attack.* Here, the first non-positive coefficient of $(1-t)^{-41}(1-t^2)^{40}(1-t^4)^{40}$ happens at $D_{XL} = 14$. So for $q > 14$, $T = 3245372870670$. We find solving a q-semiregular system of 40 quadratics and 40 quartics over GF(q) XL-Wiedemann to take $\lesssim 2^{95}\mathtt{m}$ (about 2^{100} cycles). Guessing variables may cut the memory requirement and aid parallelization but does not decrease the number of multiplications. This means that $\mathtt{QUAD}(q, 40, 40)$ is considerably below a 128-bit security level, but we know of no attack below the 80-bit security level considered in [7].

Observation: Unless it is possible to run the elimination in $\mathbf{F_4}$-$\mathbf{F_5}$ with a speed that matches the sparse matrix solvers, these more sophisticated methods would be dragged down by the amount of memory and memory operations required. E.g., against $\mathtt{QUAD}(256, 20, 20)$, even though we are operating $\mathbf{F_5}$ with a lower $D_{F5} = 9$. $T_{F5} = 6906900$, and there is just too much memory needed.

Conjecture: For generic (i.e., most random) $\mathbf{P}, \mathbf{Q}$ and $\mathbf{y}_0, \mathbf{y}_1$, $\mathbf{P}(\mathbf{x}) = \mathbf{y}_0$ and $\mathbf{P}(\mathbf{Q}(\mathbf{x})) = \mathbf{y}_1$ is 256-semiregular.

We cannot produce a hard proof, but we include a set of our test results (all using i386 code on a P4) as Table 2. The timings for $n = 6$ all the way up to $n = 15$ are consistent with our theoretical predictions. Furthermore, the timings for a $\mathtt{QUAD}(256, n, n)$ attack are identical to the timings for n randomly generated quadratics plus n randomly generated quartics in n variables.

n	6	7	8	9	10	11	12	13	14	15
D	6	6	7	7	7	7	8	8	8	8
C_{XL}	1.22	4.49	$6.08 \cdot 10$	$2.29 \cdot 10^2$	$7.55 \cdot 10^2$	$2.30 \cdot 10^3$	$5.12 \cdot 10^4$	$1.54 \cdot 10^5$	$4.39 \cdot 10^5$	$1.17 \cdot 10^6$
$\lg C_{XL}$	$2.85 \cdot 10^{-1}$	2.17	5.92	7.84	9.56	$1.12 \cdot 10$	$1.56 \cdot 10$	$1.72 \cdot 10$	$1.87 \cdot 10$	$2.02 \cdot 10$
T	$9.24 \cdot 10^2$	$1.72 \cdot 10^3$	$6.44 \cdot 10^3$	$1.14 \cdot 10^4$	$1.94 \cdot 10^4$	$3.28 \cdot 10^4$	$1.26 \cdot 10^5$	$2.03 \cdot 10^5$	$3.20 \cdot 10^5$	$4.90 \cdot 10^5$
aTm	49	65	96	120	147	177	245	288	335	385
clks	28.8	23.5	15.3	14.6	13.6	12.1	13.1	12.9	12.8	12.7

Table 2. Direct XL-Wiedemann attack on $\mathtt{QUAD}(256, n, n)$, MS C++ 7; P-D 3.0GHz, 2GB DDR2-533

T: #monomials, aTm: average terms in a row, clks: number of clocks per multiplication.

4.3 Asymptotics for Attacking $\mathtt{QUAD}(q, n, n)$ Directly (q large)

$$D_{XL} = \min\left\{D : [t^D]\ \frac{1}{(1-t)^{n+1}}\ \left((1-t^2)(1-t^4)\right)^n < 0\right\},$$

gives the degree D of the eventual XL operation. Here, $D = (w + o(1))n$, and $w \approx$ smallest positive zero w of $\oint \frac{(1-z^2)^n(1-z^4)^n}{(1-z)^{n+1} z^{wn+1}} dz = \oint \frac{dz}{z(1-z)} \left(\frac{(1+z)(1-z^4)}{z^w}\right)^n$ (cf. [28]). As usual in such situations, the expression on the RHS can only vanish when the saddle point equation of this integral has double roots (a "monkey saddle"), and here the saddle point equation is $(w-5)z^4 + z^3 - z^2 + z - w = 0$ which has a double root when w is very close to 0.2 (actually ≈ 0.200157957). So $\lg T/n \to (1+w)\lg(1+w) - w \lg w \approx 0.78$. Assuming Wiedemann solving to continue without a problem, this means that the system will be solvable in $2^{1.56n+o(n)} \times$ (polynomial in n) asymptotically.

Suppose we use, in addition to the degree-2 and degree-4 equations, the degree-8 equations $\mathbf{P}(\mathbf{Q}(\mathbf{Q}(\mathbf{x}_0))) = \mathbf{y}_2$. This reduces the final degree to

$$D_{XL} = \min\left\{D : [t^D]\ \frac{1}{(1-t)^{n+1}}\ \left((1-t^2)(1-t^4)(1-t^8)\right)^n < 0\right\},$$

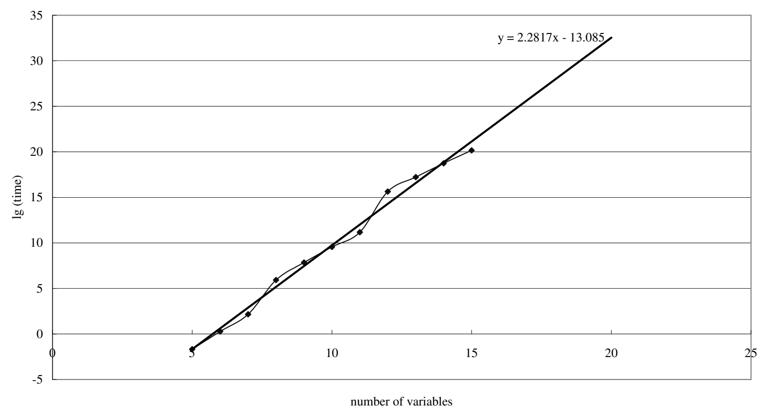

Fig. 1. Logarithmic growth of direct QUAD attack time over GF(256), optimal D

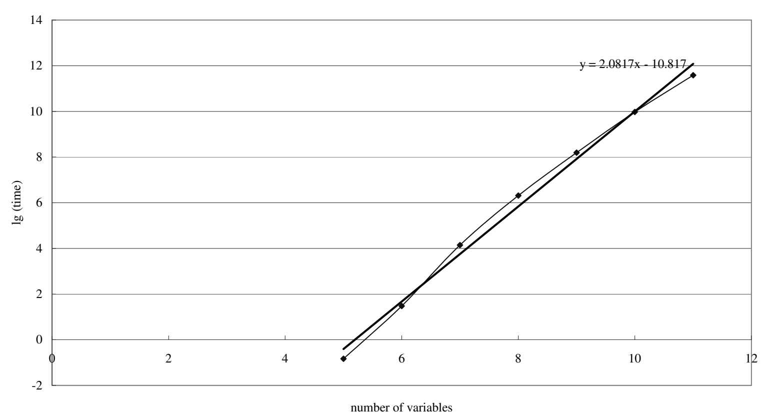

Fig. 2. Logarithmic growth of direct QUAD attack time over GF(256), $D = 7$

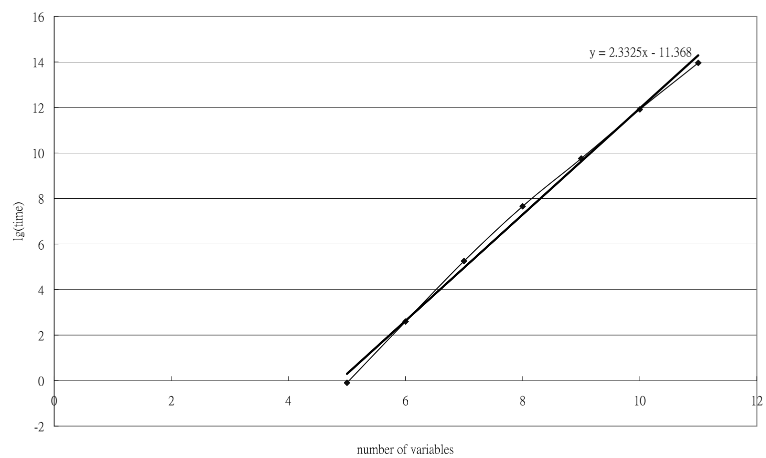

Fig. 3. Logarithmic growth of direct `QUAD` attack time over GF(256), $D = 8$

differentiating with respect to z and then finding the root of the discriminant gets us $w \approx 0.1991777370$ which confirms our suspicion that higher-degree equations provide very little improvement in the cost of the attack for large q.

4.4 Asymptotics for Attacking QUAD$(2, n, n)$ Directly

For $q = 2$ there is noticeably more benefit from using higher-degree equations and from guessing variables. We combine n quadratic equations $\mathbf{P}(\mathbf{x}_0) = \mathbf{y}_0$, n quartic equations $\mathbf{P}(\mathbf{Q}(\mathbf{x}_0)) = \mathbf{y}_1$, and so on through n equations $\mathbf{P}(\mathbf{Q}^\ell(\mathbf{x}_0)) = \mathbf{y}_\ell$ of degree 2^ℓ. The degree of operation of XL is

$$D_{XL} \gtrsim \min \left\{ D : [t^D]\ \frac{(1+t)^n}{(1-t)}\ \left((1+t^2)(1+t^4)\cdots(1+t^{2^\ell}) \right)^{-n} < 0 \right\},$$

where equality holds for 2-semi-regular systems, and which for $\ell \to \infty$ becomes $D_{XL} = \min \left\{ D : [t^D]\ (1-t)^{n-1}(1+t)^{2n} < 0 \right\}$. If we guess at f variables, then we have $D_{FXL} = \min \left\{ D : [t^D]\ (1-t)^{n-1}(1+t)^{2n-f} < 0 \right\}$. Let $c = f/n$. We would like to find a c for which the running time is smallest.

Proposition 4.5 ([4]) *We have the following asymptotics*

1. *If $f/m \to \alpha$ as $f, m \to \infty$, then*

$$D_{reg} = \min \left\{ D : [t^D]\ (1-t)^f (1+t)^m < 0 \right\} \sim \left(\frac{1}{2} - \sqrt{\alpha} + \frac{\alpha}{2} \right) m + o(m^{1/3}).$$

2. If $D/n \to w$, then $\lg T/n \to -w \lg w - (1-w)\lg(1-w)$.

Using the above, we see that $D_{FXL} \sim \left(\frac{1}{2} - \sqrt{2-c} + \frac{2-c}{2}\right) n$, hence for XL-Wiedemann, as $n \to \infty$, the limiting exponential factor $\frac{\lg C_{FXL}}{n} \to$

$$2\Big((2-c)\lg(2-c) - (\tfrac{3-c}{2} - \sqrt{2-c})\lg(\tfrac{3-c}{2} - \sqrt{2-c}) - (\tfrac{1-c}{2} + \sqrt{2-c})\lg(\tfrac{1-c}{2} + \sqrt{2-c})\Big) + c.$$

Due to constraints in asymptotic analysis which are too lengthy to discuss here, this expression is only guaranteed to hold for small c and may not continue to hold as c goes up. But it is clear that asymptotically, FXL is the correct approach here. A rough estimate is that the security under the direct attack is $2^{1.02n}$ with XL, and $2^{0.86n}$ with best FXL.

4.6 Which Field Is Best?

Our analysis suggests that after taking into account all known improvements, to achieve any particular level of security against known attacks, $q = 2$ needs considerably more variables than a large q, the ratio being about $1.56/0.86 \approx 1.81$. The `QUAD` user can try to save time by increasing q and reducing the number of variables. Our current impression is that $q = 256$ is not a good choice: the extra cost of arithmetic in $\mathrm{GF}(q)$ offsets the smaller number of variables and polynomial coefficients to be manipulated. However, $q = 16$ or $q = 4$ may be better. These estimates will have to be updated if faster attacks are discovered.

4.7 The State of System-Solving Cryptanalysis

We have shown that Lazard-Faugère solvers, particularly XL with a sparse matrix solver, can be very useful in attacking generic systems, or at least systems that have generic or randomly created elements. In some cases such simple approaches are more useful than the really sophisticated variations.

There is a problem we ran into when trying to organize a medium-scale parallel cluster to help crack a `QUAD` instance. It is desirable to partition the matrix because memory capacity is always a problem. Yet in Block Lanczos and Block Wiedemann each computer must hold a copy of the entire matrix. If we do the naive parallelization, then the communications cost goes up squarely as the cluster size but the efficiency gain only linearly. This will take a good tuning.

5 Unprovable Parameters for QUAD

5.1 Overview: Attacking the Underlying Hard Problem

The `QUAD` paper [7, page 110] states that the security of `QUAD` "is provably reducible to the intractability of the MQ problem [15], which consists of finding a solution (if any) to a multivariate quadratic system of m quadratic equations in n

variables over a finite field GF(q), typically GF(2).” Here $m = n+r$. The quoted statement means that there is a proof that *assumes* that $\mathcal{MQ}$ is intractable and *concludes* that `QUAD` is secure.

We emphasize that the intractability of the $\mathcal{MQ}$ problem is not a theorem; it is a hypothesis. For some parameter choices, the hypothesis is plausible. For other parameter choices, the hypothesis is false. For example, we show in this section how to break 80 quadratic equations in 40 variables over GF(16).

For parameters where the associated $\mathcal{MQ}$ instance is shown to be not “hard,” any attempt to show provable security starting from the hardness of the $\mathcal{MQ}$ problem becomes hopeless; hence our terminology “unprovable.” Note that unprovable parameters can be analyzed even before a proof has been claimed or written down; unprovability is deduced purely from the parameter choices for the underlying “hard problem.”

5.2 Example: QUAD(256, 20, 20)

We return to `QUAD`$(256, 20, 20)$ as an illustrative example. What the quoted statement says in this case is that the security of `QUAD`$(256, 20, 20)$ follows from the difficulty of solving 40 quadratic equations in 20 variables over GF(256).

Is solving 40 random quadratic equations in 20 variables over GF(256) actually difficult? No, not at all! We can compute from Prop. 3.5 that $D_{XL} = 5$ and $T = 53130$. The maximum number of terms per equation is $k \lesssim 231$, so on a P4, $C_{XL} \approx 9 \times 10^{12} \lesssim 2^{45}$. To summarize: XL-Wiedemann takes $< 2^{45}$ cycles, which is only a few hours on a decent computer.

To verify our estimates, we carried out this computation the same evening that we began to study the security of `QUAD`. XL-Wiedemann solved the equations on schedule. The fairly mature $\mathbf{F_4}$ in MAGMA version 2.12 did the same job in about a quarter of the time, presumably using more memory.

Of course, after establishing the breakability of `QUAD`$(256, 20, 20)$ in Section 4, we could apply the contrapositive of the “provable security” of `QUAD` to deduce the breakability of the corresponding $\mathcal{MQ}$ system. However, this circuitous approach would be considerably harder to verify than a direct attack on $\mathcal{MQ}$. We also observe a significant gap between the cost of the $\mathcal{MQ}$ attack and the cost of the `QUAD` attack. Showing unprovability of a set of `QUAD` parameters is easier than showing breakability.

5.3 Example: QUAD(16, 40, 40)

For `QUAD`$(16, 40, 40)$, we can verify (cf. Prop. 3.5) $D = 8$, $T = 377348994$, and $k \lesssim 861$. Since the storage will come to about 800 GB among a small cluster, we will assume AMD64 or a similar architecture, and it will take about 8 cycles a multiplication before tuning, and a total of $3 \times 8 \times 377348994^2 \times 861 \approx 3 \times 10^{20} \lesssim 2^{72}$ cycles. With a little work, such as unrolling by hand the critical loop, we can expect to cut this down to 2^{71} on an Opteron.

These attacks are considerably below the specified 2^{80} security level. We conclude that `QUAD`$(16, 40, 40)$ is unprovable.

5.4 Examples: QUAD(2, 160, 160), (2, 256, 256), (2, 350, 350)

For QUAD$(2, 160, 160)$: $D = 13$ (this time cf. Prop. 3.4), but this time $T = 480198915703266914 9$, and $k \lesssim 12881$ which makes it rather difficult to fit everything into memory. An optimistic 6-cycles-per-multiplication hypothesis, ignoring space problems, leads to an estimate of approximately 2^{140} cycles.

For QUAD$(2, 256, 256)$: $D = 19$, $T = 25707968047799666061215057489$ or $\lesssim 2^{94}$. $C_{XL} \approx 2^{205}\mathfrak{m}$. For QUAD (2,350,350), $D = 24$, $T \lesssim 2^{123}$. $C_{XL} \lesssim 2^{263}\mathfrak{m}$. [In the astronomical realm, a multiplication may take rather more than 6 cycles.]

6 Proven and Unproven Parameters for QUAD

There are several limitations on what was actually proven in [7]. We focus on one limitation, namely the lack of tightness in [7, Theorem 4].

The theorem does not claim that a QUAD attack implies an $\mathcal{MQ}$ attack *with the same efficiency*. It says that a QUAD attack implies an $\mathcal{MQ}$ attack *with a bounded loss of efficiency*: specifically, if λn bits of output from QUAD$(2, n, r)$ can be distinguished from uniform with advantage ϵ in time T, then a random $\mathcal{MQ}$ system of $n + r$ equations in n variables over GF(2) can be solved with probability $2^{-3}\epsilon/\lambda$ in time

$$T' \leq \frac{2^7 n^2 \lambda^2}{\epsilon^2} \left(T + (\lambda + 2)T_S + \log\left(\frac{2^7 n \lambda^2}{\epsilon^2}\right) + 2 \right) + \frac{2^7 n \lambda^2}{\epsilon^2} T_S,$$

where $T_S :=$ time to run one block of QUAD$(2, n, r)$.

How could we conclude the security of QUAD—a lower bound on T, such as $T \geq 2^{80}$—from this theorem? A minor point is that the theorem would need to be extended to all q, not just $q = 2$, and proven; we will make the (questionable) assumption that this is done, and that it does not produce worse time bounds. The major point is that we would need to assume a much larger lower bound on T', compensating for (among other things) a factor of $2^{10}n^2\lambda^3/\epsilon^3$. For example, as in [7] let's accept ϵ as large as 0.01, and let's put $L = \lambda n = 2^{40}$. The extra factor is then $2^{150}/n$. The theorem cannot conclude $T \geq 2^{80}$ without assuming that $T' \geq 2^{230}/n$. Let's check this against cases studied in Section 5:

- QUAD$(256, 20, 20)$: Unproven. Our (computer-verified) estimate is $T' \leq 2^{45}$.
- QUAD$(16, 40, 40)$: Unproven, we estimate $T' \leq 2^{71}$.
- QUAD$(2, 350, 350)$: *Proven*, if there are no better $\mathcal{MQ}$ attacks. We estimate $T' \approx 2^{263}\mathfrak{m}$. A 2^{80} distinguishing attack would lead to an $\approx 2^{221}\mathfrak{m}$ expected-time solution. Note $T' \approx 2^{222}\mathfrak{m}$ for QUAD$(2, 320, 320)$, which should suffice.
- QUAD$(2, 160, 160)$: Unproven ($T' \leq 2^{140}$, [7] specified this as unproven).
- QUAD$(2, 256, 256)$: *Proven* for the parameters [7] $L = 2^{22}$, $\epsilon = 0.01$, if there are no better $\mathcal{MQ}$ attacks. We estimate $T' \approx 2^{205}\mathfrak{m}$ where we need $2^{168}\mathfrak{m}$.

Why did [7] overestimate n? *The $N^{2.376}$ formula discounts an expected big coefficient, compensating for our improvement to $N^{2+\epsilon}$ and then some.*

The bottom line is that *all three* QUAD parameter choices used for speed reports in the QUAD talk at Eurocrypt 2006 are unproven, although the gap in the QUAD$(2, 160, 160)$ case might be closed by a tighter security proof.

References

1. G. Ars, J.-C. Faugère, H. Imai, M. Kawazoe, and M. Sugita. Comparison between XL and Gröbner Basis algorithms. In AsiaCrypt [25], pages 338–353.
2. M. Bardet. *Étude des systèmes algébriques surdéterminինés. Applications aux codes correcteurs et á la cryptographie.* PhD thesis, Université Paris VI, 2004.
3. M. Bardet, J.-C. Faugère, and B. Salvy. On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.
4. M. Bardet, J.-C. Faugère, B. Salvy, and B.-Y. Yang. Asymptotic expansion of the degree of regularity for semi-regular systems of equations. In P. Gianni, editor, *MEGA 2005 Sardinia (Italy)*, 2005.
5. C. Berbain, O. Billet, and H. Gilbert, Efficient implementations of multivariate quadratic systems. Workshop record distributed at 13th Annual Workshop on Selected Areas in Cryptography, August 2006.
6. C. Berbain and H. Gilbert. On the security of IV dependent stream ciphers. Workshop record distributed at 14th Annual Workshop on Fast Software Encryption, March 2007.
7. C. Berbain, H. Gilbert, and J. Patarin. QUAD: A practical stream cipher with provable security. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2006.
8. L. Blum, M. Blum, and M. Shub. Comparison of two pseudo-random number generators. pages 61–78, New York, 1983. Plenum Press.
9. B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, Innsbruck, 1965.
10. Computational Algebra Group, University of Sydney. *The MAGMA Computational Algebra System for Algebra, Number Theory and Geometry.* `http://magma.maths.usyd.edu.au/magma/`.
11. N. Courtois. Algebraic attacks over $GF(2^{\mathrm{k}})$, application to HFE challenge 2 and Sflash-v2. In *Public Key Cryptography — PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 201–217. Feng Bao, Robert H. Deng, and Jianying Zhou (editors), Springer, 2004. ISBN 3-540-21018-0.
12. N. T. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Bart Preneel, editor, Springer, 2000. Extended Version: `http://www.minrank.org/xlfull.pdf`.
13. N. T. Courtois and J. Patarin. About the xl algorithm over gf(2). In *The Cryptographer's Track at RSA Conference 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2003.
14. N. T. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Yuliang Zheng, editor, Springer, 2002.
15. C. Diem. The xl-algorithm and a conjecture from commutative algebra. In AsiaCrypt [25], pages 323–337. ISBN 3-540-23975-8.
16. J. Ding, J. Gower, and D. Schmidt. *Multivariate Public-Key Cryptosystems.* Advances in Information Security. Springer, 2006. ISBN 0-387-32229-9.
17. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139:61–88, June 1999.

18. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *International Symposium on Symbolic and Algebraic Computation — ISSAC 2002*, pages 75–83. ACM Press, July 2002.
19. J.-C. Faugère and A. Joux. Algebraic cryptanalysis of Hidden Field Equations (HFE) using Gröbner bases. In *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Dan Boneh, editor, Springer, 2003.
20. M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979. ISBN 0-7167-1044-7 or 0-7167-1045-5.
21. N. Koblitz and A. Menezes. Another look at "provable security". Cryptology ePrint Archive, Report 2004/152, 2004. `http://eprint.iacr.org/`.
22. N. Koblitz and A. Menezes. Another look at "provable security". ii. In R. Barua and T. Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 148–175. Springer, 2006.
23. D. Lazard. Gröbner-bases, Gaussian elimination and resolution of systems of algebraic equations. In *EUROCAL 83*, volume 162 of *Lecture Notes in Computer Science*, pages 146–156. Springer, March 1983.
24. F. S. Macaulay. *The algebraic theory of modular systems*, volume xxxi of *Cambridge Mathematical Library*. Cambridge University Press, 1916.
25. Pil Joong Lee, editor. *Advances in Cryptology — ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*. Springer, 2004. ISBN 3-540-23975-8.
26. H. Raddum and I. Semaev. New technique for solving sparse equation systems. Cryptology ePrint Archive, Report 2006/475, 2006. `http://eprint.iacr.org/`.
27. C. Wolf and B. Preneel. Taxonomy of public key schemes based on the problem of multivariate quadratic equations. Cryptology ePrint Archive, Report 2005/077, 12th of May 2005. `http://eprint.iacr.org/2005/077/`, 64 pages.
28. R. Wong. *Asymptotic approximations of integrals*, volume 34 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001. Corrected reprint of the 1989 original.
29. B.-Y. Yang and J.-M. Chen. All in the XL family: Theory and practice. In *ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 67–86. Springer, 2004.
30. B.-Y. Yang and J.-M. Chen. Theoretical analysis of XL over small fields. In *ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 277–288. Springer, 2004.