XTR, a public key cryptosystem

Lucie van der Logt

juli 2001

Contents

1 Cryptography

In this section a few important cryptographic issues that are needed later on will be explained.

1.1 Public key cryptography

Cryptography makes it possible to send over messages safely. It has three main purposes:

1. *confidentiality:* users of cryptosystems want to be sure that they can send messages safely. In case someone overhears the conversation, they want to be sure that the message cannot be understood.
2. *non-repudiation:* as receiver you want to be able to verify that nobody has tampered with the message. This means that you want to check if the message that arrived, is the same message as the one that was sent.
3. *authentification:* as receiver you want to be able to verify that the message really was sent by the person you think was it sent by.

There are two kinds of cryptography: private key cryptography and public key cryptography. In the first kind of cryptography it is necessary for the parties who want to exchange secret information agree upon a secret key beforehand. Public key cryptography made it possible to exchange keys without meeting. In public key cryptosystems every user has a public key (that may be known by everybody) and a private key (only known by himself). These public key cryptosystems are based on so called *one-way functions.*

Definition 1.1.1.
A one-to-one function $f : X \to Y$ is called a one-way function *if it is "easy" to compute $f(x)$, for all $x \in X$, but "hard" to compute x such that $f(x) = y$ for any $y \in Y$ for which such an x exists.*

In cryptography usely the assumption is made that the message and the keys are all integers or elements of a finite field. So for X and Y in definition 1.1.1 you usely take $\mathbb{Z}$or a finite field. In this thesis all cryptosystems described assume that the messages and the keys are elements of a finite field $GF(p^n)$.

1.2 Trapdoor functions

The one-way functions that can be used for cryptographic applications need an extra property. They must have a "trapdoor". This means that with some information it becomes easy to compute x such that $f(x) = y$ for $y \in Y$ for which such a x exists. There are several functions that can be used as trapdoor function. These are all based on problems that are regarded to as "difficult to solve". These problems are problems like factorization of large numbers and the discrete logarithm problem for groups (DLP) The cryptosystem described in this thesis is based on the DLPin the multiplicative subgroups of finite fields. This is the problem that given a $g \in GF(p^n)^*$ and g^x it's hard to compute x. As long as the prime p or the order of g stay small, the x can be computed by brute force methods, but for large primes this takes too long.

1.3 Kinds of public key cryptosystems

There are all kinds of possible cryptosystems. Some of them will be shown in this section.

1.3.1 Key agreement

To be able to use a private key cryptosystem the users have to agree upon a common private key. They can use a public key cryptosystem to do so. The Diffie-Hellman protocol is a protocol for key distribution. It allows two people to agree upon an shared secret key over an open channel without meeting. Suppose Alice wants to send a secret message to Bob using a private key cryptosystem. Let p, q be primes, $n \in \mathbb{N}$ and $g \in GF(p^n)$ such that the order of g is dividing q. These primes p and q should be chosen large enough so that we can assume that the DLP cannot be broken.
These parameters can all be public.
Alice and Bob want to select a secret key k. They can do that by following algorithm 1.3.1:

Algorithm 1.3.1 (Diffie-Hellman key agreement).

1. *Alice selects at random $a \in \mathbb{Z}$ such that $1 < a < q-2$ and computes g^a. She sends g^a to Bob.*
2. *Bob selects at random $b \in \mathbb{Z}$ such that $1 < b < q-2$ and computes g^b. He sends g^b to Alice.*
3. *Alice computes $(g^b)^a = g^{ab}$.*
4. *Bob computes $(g^a)^b = g^{ab}$.*

The g^{ab} computed by Alice and Bob can now be used as secret key in a private key system. Suppose someone followed the conversation then it's impossible for this person to compute g^{ab}, because he only knows g^a and g^b and it's considered impossible to compute a or b out of these (DLP).

1.3.2 Encryption

There are also public key systems with which you can directly encrypt a message. Suppose Alice has as public key (p, q, g, g^k) in which p and q are large primes, $g \in GF(p^n)$ such that the order of g is dividing q and k a secret integer between 1 and $q-2$. Bob wants to send her a message M. He can follow the ElGamal algorithm 1.3.2 to do this safely.

Algorithm 1.3.2 (ElGamal encryption).

1. *Bob selects at random $b \in \mathbb{Z}$ such that $1 < b < q-2$ and computes g^b and $(g^k)^b = g^{bk}$.*
2. *Bob computes $g^{bk}M$ and sends $(g^b, g^{bk}M)$ to Alice.*
3. *Alice computes $(g^b)^{-k} = g^{-bk}$ out of g^b and k.*

4. *Alice computes M out of $g^{-bk}M$ and g^{bk}.*

This system is also based on the safety of the DLP because someone who eavesdrops the communication needs to compute k out of g^k to be able to decode the message. To improve this system Alice and Bob can agree upon a private key system to encrypt M with private key g^{bk} instead of sending $g^{bk}M$.

1.3.3 Hashing

With the above algorithms it is possible to encrypt a message and send it over safely. But it does not ensure non-repudiation and doesnot make authentification possible. Non-repudiation can be guaranteed by using *hashing.*

Definition 1.3.3.
A function $h : X \to Y$ is called a hash-function *if:*

- *it is "easy" to compute $h(x)$ for all $x \in X$*
- *it's "hard" to find x and x' such that $h(x) = h(x')$*
- *it's "hard" to compute x such that $h(x) = y$ for $y \in Y$ for which such an x exists.*

The way these functions can be used is shown in algorithm 1.3.4. Alice wants to send a message M to Bob and they want to make sure that nobody tampers with the message on the way. Suppose h is a hash-function and Alice and Bob have agreed upon an encryption algorithm for example as in 1.3.2.

Algorithm 1.3.4 (Hashing).

1. *Alice computes (before sending) $h(M)$.*
2. *She encrypts the message M with the encryption algorithm they agreed on and sends the encrypted message together with $h(M)$ to Bob.*
3. *Bob decrypts the message.*
4. *Bob computes $h(M)$ and checks if it is the same as the $h(M)$ send by Alice.*

If the $h(M)$'s are the same, nothing happened on the way. Because, suppose someone tampered the message, then he/she would be able to compute the h of the tampered with message, but then Bob would notice that it is different. Because of the choice of h the eavesdropper is not able to tamper with the message without affecting the value of h.

1.3.4 Digital signature

To assure authentification one can use a so called digital signature. A digital signature is something you add to your message so that the receiver can verify that you are really the one who sent the message. An example of an algorithm to sign a message is the Nyberg-Rüppel algorithm. This algorithm combines both hashing and a digital signature. Again Alice and Bob have agreed upon an encryption algorithm, for example as in 1.3.2 and h is a hash-function. Alice's public key is (p, q, g, g^k).

Algorithm 1.3.5 (Nyberg-Rüppel signature).

1. *Alice selects at random* $a \in \mathbb{Z}$ *such that* $1 < a < q - 2$ *and computes* g^a.
2. *Alice appends her "signature" for example:* "This message comes from Alice." *to the message she wants to send.*
3. *Alice encrypts the message* M *with* g^a *as the key, which leads to an encrypted message* E.
4. *Alice computes* $h(E)$ *and* $s = (k \cdot h(E) + a) \bmod q$, *and sends Bob* E *and* s.
5. *Bob computes* $h(E)$, g^a *and* $(g^k)^{-h(E)}$. *With those he computes* $g^s g^{-h(E)k} = g^a$.
6. *He uses* g^a *to decrypt* E.
7. *He checks wether the signature is the signature they agreed upon.*

If the text that (Alice appended and) Bob decrypts is the same as the text they agreed on to append, then Bob can be sure that the message comes from Alice. This follows from the fact that Alice is the only one who can compute s, because she's the only one who knows k. (Here again is used that it's hard to compute k out of the public key g^k (DLP)). Suppose someone else tries to send a message as if it comes from Alice. Then Bob gets the wrong s and the message he gets from the decryption will not contain the text they agreed on, but something incomprehensible.

In the previous subsections examples are given of algorithms that can be used for cryptographic purposes. Of course there are many more possibilities and all these possibilities can again be combined to get a better system that guarantees safe communication.

2 XTR

In this section the XTR public key system will be explained. It's a new method from Arjen K. Lenstra and Eric R. Verheul based on the discrete logarithm problem.

2.1 Introduction

XTR stands for **E**fficient and **C**ompact **S**ubgroup **T**race **R**epresentation. It is based on the Diffie-Hellman protocol using a subgroup of $GF(p^6)^*$ for a prime p. This subgroup is generated by $g \in GF(p^6)^*$ with order dividing another prime q.

Lemma 2.1.1. *Let $p \equiv 2$ mod 3 be a prime such that there exists a prime $q > 6$ with $q \mid p^2 - p + 1$. Let $g \in GF(p^6)^*$ with $o(g) = q$. Then g is not contained in a proper subfield of $GF(p^6)$.*

Proof. The only subfields of $GF(p^6)$ are $GF(p)$, $GF(p^2)$ and $GF(p^3)$.
Suppose $g \in GF(p)$ then $q \mid p-1$ and so, because $q \mid p^2 - p + 1$, $q \mid p^2$ and thus $q \mid p$. But that would mean that $q = 1$ which is in contradiction with $q > 6$.
Suppose $g \in GF(p^2)$ then $q \mid p^2 - 1$ and thus $q \mid p - 1$ or $q \mid p + 1$. $q \mid p - 1$ leads to contradiction because then g would be in $GF(p)$. So $q \mid p + 1$. Because also $q \mid p^2 - p + 1$ it follows that $q \mid 2p + 1$ and thus $q \mid p$. Contradiction.
Suppose $g \in GF(p^3)$ then $q \mid p^3 - 1$ so $q \mid p - 1$ or $q \mid p^2 + p + 1$. $q \mid p - 1$ isn't possible so $q \mid p^2 + p + 1$. Thus $q \mid 2p$. But then $q \mid 2$ or $q \mid p$ which again leads to contradiction.

□

For the rest of this chapter we'll take such p and q because then we'll truly work in $GF(p^6)$ and not in some subfield. In the normal Diffie-Hellman protocol you would need a representation of $GF(p^6)$ to represent g and its powers. The idea of XTR is that you don't need to use that, but you can represent g by its trace and $Tr(g)$ is an element of $GF(p^2)$. With this new representation you can achieve a reduction in the computational costs and communication time. In the remaining part of this chapter we'll show how to do this.

2.2 Representation of $GF(p^2)$ over $GF(p)$

Before explaining how to use $Tr(g)$ instead of g it's first necessary to have an efficient representation of $GF(p^2)$ over $GF(p)$. Let α be a zero of $X^2 + X + 1$, an irreducible polynomial over $GF(p)$. Then α^p is also a zero and α and α^p form an optimal normal basis for $GF(p^2)$. This means that for every $x \in GF(p^2)$ there exist x_1 and $x_2 \in GF(p)$ with $x = x_1\alpha + x_2\alpha^p$.
Because $\alpha^3 = \alpha\alpha^2 = \alpha(-\alpha - 1) = -\alpha^2 - \alpha = \alpha + 1 - \alpha = 1$, it follows that $\alpha^i = \alpha^{i \bmod 3}$. Therefore and because $p \equiv 2 \bmod 3$ for every $x \in GF(p^2)$ there exist x_1 and $x_2 \in GF(p)$ with $x = x_1\alpha + x_2\alpha^2$.
The following lemma says how much the costs of multiplications in $GF(p^2)$ are.

Lemma 2.2.1. *Let $x, y, z \in GF(p^2)$. Then:*

(i) Computing x^p takes 0 multiplications in $GF(p)$.

(ii) Computing x^2 takes 2 multiplications in $GF(p)$.

(iii) Computing xy takes 3 multiplications in $GF(p)$.

(iv) Computing $xz - yz^p$ takes 4 multiplications in $GF(p)$.

Proof.

(i) $x^p = (x_1\alpha + x_2\alpha^2)^p = x_1^p\alpha^p + x_2^p\alpha = x_1\alpha^p + x_2\alpha$ so computing x^p takes just a transposition.

(ii) $x^2 = (x_1\alpha + x_2\alpha^2)^2 = x_2(x_2 - 2x_1)\alpha + x_1(x_1 - 2x_2)\alpha^2$ and this costs 2 multiplications in $GF(p)$. (Multiplying by 2 is an addition.)

(iii) $xy = (x_1\alpha + x_2\alpha^2)(y_1\alpha + y_2\alpha^2) = (x_2y_2 - (x_1y_2 + x_2y_1))\alpha + (x_1y_1 - (x_1y_2 + x_2y_1))\alpha^2$. $x_1y_2 + x_2y_1$ can be computed as follows: $x_1y_2 + x_2y_1 = (x_1 + x_2)(y_1 + y_2) - x_1y_1 - x_2y_2$. So xy can be computed by computing $(x_1 + x_2)(y_1 + y_2)$, x_1y_1 and x_2y_2, that is 3 multiplications in $GF(p)$.

(iv) $xz - yz^p = (x_1\alpha + x_2\alpha^2)(z_1\alpha + z_2\alpha^2) - (y_1\alpha + y_2\alpha^2)(z_1\alpha + z_2\alpha^2)^p = (z_1(y_1 - y_2 - x_2) + z_2(x_2 - x_1 + y_2))\alpha + (z_1(x_1 - x_2 + y_1) + z_2(y_2 - y_1 - x_1))\alpha^2$ and this takes 4 multiplications in $GF(p)$.

□

2.3 Computing $Tr(g^n)$ out of $Tr(g)$

For the cryptographic applications it will be necessary to be able to compute $Tr(g^n)$ out of $Tr(g)$. How this can be done will be explained in this section.

Theorem 2.3.1.
The minimal polynomial of g^n over $GF(p^2)$ is $X^3 - Tr(g^n)X^2 + Tr(g^n)^pX - 1$. Specially the minimal polynomial of g over $GF(p^2)$ is $X^3 - Tr(g)X^2 + Tr(g)^pX - 1$.

Proof.
Let $f(X)$ be the minimal polynomial of g^n. That means that $f(g^n) = 0$. Furthermore $f((g^n)^{p-1}) = f((g^n)^{p^2}) = (f(g^n))^{p^2} = 0$ because the characteristic is p and $o(g^n) \mid q$. Also $f((g^n)^{-p}) = f((g^n)^{(p-1)^2}) = f((g^n)^{p^4}) = f((g^n))^{p^4} = 0$. And because g^n has two conjugates over $GF(p^2)$ follows that those conjugates are $(g^n)^{p-1}$ and $(g^n)^{-p}$. Therefore $f(X) = (X - g^n)(X - (g^n)^{p-1})(X - (g^n)^{-p}) = X^3 - Tr(g^n)X^2 + Tr(g^n)^pX - 1$. □

Definition 2.3.2.
For $n \in \mathbb{N}$ define $c_n := Tr(g^n)$ and let $F(c_n, X) := X^3 - c_nX^2 + c_n^pX - 1$ be the minimal polynomial of g^n. Furthermore let $g_{n,1} := (g^n)^{p-1}$ and $g_{n,2} := (g^n)^{-p}$ be the conjugates of $g_{n,0} := g^n$.

The question is now how to calculate c_n out of c_1. To answer that we first need a few lemmas that tell us how to calculate with c_n.

Lemma 2.3.3.

(i) $g_{n,0} \cdot g_{n,1} \cdot g_{n,2} = 1$ *for* $n \in \mathbb{Z}$.

(ii) $g_{n,0} \cdot g_{n,1} + g_{n,0} \cdot g_{n,2} + g_{n,1} \cdot g_{n,2} = c_{-n}$ *for* $n \in \mathbb{Z}$.

(iii) $c_{-n} = c_{np} = c_n^p$ *for* $n \in \mathbb{Z}$.

(iv) $c_{u+v} = c_u c_v - c_v^p c_{u-v} + c_{u-2v}$ *for* $u, v \in \mathbb{Z}$.

(v) $c_{2n} = c_n^2 - 2c_n^p$.

(vi) $c_{n+2} = c_1 c_{n+1} - c_1^p + c_{n-1}$.

(vii) $c_{2n+1} = c_{n+1} c_n - c_1 c_n^p + c_{n-1}^p$.

Proof.

(i) $g_{n,0} \cdot g_{n,1} \cdot g_{n,2} = -[F(c_n, X)]_{X=0} = 1$.

(ii) From $g_{n,0} \cdot g_{n,1} \cdot g_{n,2} = 1$ it follows that $(g_{n,0})^{-1} = g_{n,1} \cdot g_{n,2}$ and the same for $g_{n,1}$ and $g_{n,2}$. Thus $g_{n,0} \cdot g_{n,1} + g_{n,0} \cdot g_{n,2} + g_{n,1} \cdot g_{n,2} = (g_{n,2})^{-1} + (g_{n,1})^{-1} + (g_{n,0})^{-1} = g_{-n,2} + g_{-n,1} + g_{-n,0} = c_{-n}$.

(iii) $g_{n,0} = (g_{n,1})^{-p}$, $g_{n,1} = (g_{n,2})^{-p}$ and $g_{n,2} = (g_{n,0})^{-p}$. Thus $c_{-n} = g_{n,0} + g_{n,1} + g_{n,2} = (g_{n,1})^{-p} + (g_{n,2})^{-p} + (g_{n,0})^{-p} = c_{np} = c_n^p$ because the characteristic is p.

(iv) This follows by using the definitions and the computation-rules above.

(v) Use (iv) with $u = n$ and $v = n$, the fact that $c_{-n} = c_n^p$ and that $c_0 = Tr(1) = 3$.

(vi) Use (iv) with $u = n + 1$ and $v = 1$.

(vii) Use (iv) with $u = n + 1$ and $v = n$ and the fact that $c_{-n+1} = c_{n-1}^p$.

□

Because $c_{-n} = c_n^p$ and computing x^p out of x takes no multiplications, computing c_{-n} out of c_n is for free. From part (iv) and lemma 2.2.1 it follows that c_{u+v} can be computed in 4 multiplications given c_u, c_v, c_{u-v} and c_{u-2v}. So specially c_{n+2} and c_{2n-1} can be computed in 4 multiplications given c_{n-1}, c_n and c_{n+1}. c_{2n} can even be computed in 2 multiplications from c_n because p^{th} powering is for free. These analyses lead to a an algorithm to compute c_n out of c_1. Since the relations above are recursion-formulas of degree 3, the algorithm shows how to compute (c_{n-1}, c_n, c_{n+1}) out of (c_1, c_2, c_3). c_2 and c_3 can be derived from c_1 as follows:
$c_2 = c_1^2 - 2c_1^p$ because of lemma 2.3.3 (v) and $c_3 = c_1 c_2 - c_1^p c_1 + 3$ because of lemma 2.3.3 (vii).

Definition 2.3.4.
Let $S_n(g) := (c_{n-1}, c_n, c_{n+1})$.
Let $T_n(g) := S_{2n+1}(g)$.

Lemma 2.3.5.

(i) $T_{2n}(g)$ *can be computed from* $T_n(g)$ *in 8 multiplications in* $GF(p)$.

(ii) $T_{2n+1}(g)$ *can be computed from* $T_n(g)$ *in 8 multiplications in* $GF(p)$.

(iii) $S_{n+1}(g)$ *can be computed from* $S_n(g)$ *in 4 multiplications in* $GF(p)$.

Proof.

(i) $T_{2n}(g) = (c_{4n}, c_{4n+1}, c_{4n+2})$ and $T_n(g) = (c_{2n}, c_{2n+1}, c_{2n+2})$. c_{4n} follows from $T_n(g)$ with 2.3.3 (v) for $2n$. c_{4n+1} follows from $T_n(g)$ with 2.3.3 (vii) for $2n$ and c_{4n+2} follows from $T_n(g)$ with 2.3.3 (v) for $2n+1$. This takes 2+4+2 = 8 multiplications.

(ii) $T_{2n+1}(g) = (c_{4n+2}, c_{2n+3}, c_{2n+4})$. c_{4n+2} follows from $T_n(g)$ with 2.3.3 (v) for $2n+1$, c_{4n+3} follows from $T_n(g)$ with 2.3.3 (vii) for $2n+1$ and c_{4n+4} follows from $T_n(g)$ with 2.3.3 (v) for $2n+2$. This takes 2+4+2 = 8 multiplications.

(iii) $S_{n+1}(g) = (c_n, c_{n+1}, c_{n+2})$. c_{n+2} follows from $S_n(g)$ using 2.3.3(vi). This costs 4 multiplications in $GF(p)$.

□

Algorithm 2.3.6 will show how to compute $T_m(g)$ out of $T_0(g)$ for $m \in \mathbb{N}$. Algorithm 2.3.7 will then show how to compute $S_n(g)$ out of $S_2(g)$ for $n \in \mathbb{N}$ using algorithm 2.3.6. For $n < 0$ you can apply the algorithm to $-n$ and then use the fact that $c_{-n} = c_n^p$.

Algorithm 2.3.6 (Computing $T_m(g)$ out of $T_0(g)$).

1. *Write* m *binary. Say* $m = \sum_{i=0}^{r} m_i 2^i$ *with* $m_i \in \{0,1\}$ *for* $0 \leq m \leq r-1$ *and* $m_r = 1$.
2. *Compute* T_1 *by using lemma 2.3.5(ii) for* $n = 0$.
3. *Define* $a_0 := m_r$.
4. *Repeat the following step* r *times (thus for* i *from 1 to* r*):*
   Compute T_{a_i} *out of* $T_{a_{i-1}}$ *where* $a_i := 2a_{i-1} + m_{r-i}$.
5. *Now* $T_m = T_{a_r}$.

Proof.
Every computation of step 2 is possible, because either $m_{r-i} = 0$ and then you need to compute T_{2k} out of T_k which is possible according to lemma 2.3.5(i), either $m_{r-i} = 1$ and then you need to compute T_{2k+1} out of T_k which is possible according to lemma 2.3.5(ii).
$m = \sum_{i=0}^{r} m_i 2^i = 2(2(\ldots 2(2 + m_{r-1}) + m_{r-2}) + \ldots + m_1) + m_0 = a_r$, so when you've completed the algorithm you've found T_m. □

Algorithm 2.3.7 (Computing $S_n(g)$ out of $S_1(g)$).

1. *If n is even, continue the algorithm with $n-1$ and compute afterwards $S_n(g)$ with lemma 2.3.5(iii).*
2. *Find m such that $n = 2m + 1$.*
3. *Compute $T_m(g)$ using algorithm 2.3.6.*
4. *This $T_m(g)$ is $S_n(g)$.*

Proof.
The proof of the working of this algorithm follows directly from the definitions of S and T. □

The next example shows how the algorithm works.

Example 2.3.8.
To compute $S_{44}(g)$ we do the following:
44 is even so first we'll compute $S_{43}(g)$.
$43 = 2 \cdot 21 + 1$.
$21 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4$.
Starting with T_0 we make the following computations:

$T_0 \longrightarrow T_1 \longrightarrow T_2 \longrightarrow T_5 \longrightarrow T_{10} \longrightarrow T_{21}$

2.4 Computing $Tr(g^a \cdot g^{bk})$ out of $Tr(g)$

2.5 Cryptographic applications

In this paragraph it will be shown how these shorter representations can be used in the cryptosystems explained in chapter one.

2.5.1 Diffie-Hellman and XTR

Alice and Bob publiquely agree upon a p, q and $Tr(g)$. Everybody can know these. To find a private key they take the following steps (as in 1.3.1):

1. Alice selects at random $a \in \mathbb{Z}$ such that $1 < a < q-2$ and computes $S_a(g)$ using algorithm 2.3.7. She sends $S_a(g)$ to Bob.
2. Bob selects at random $b \in \mathbb{Z}$ such that $1 < a < q-2$ and computes $S_b(g)$ using algorithm 2.3.7. He sends $S_b(g)$ to Alice.
3. Alice computes $S_a(= g^{ab}$.
4. Bob computes $(g^b)^a = g^{ab}$.

3 Possible improvements to XTR

In this section some ideas about how to improve XTR are explained. The improvements are based on improving algorithm 2.3.7, the computation of $Tr(g^n)$ out of $Tr(g)$.

3.1 Addition chains

In algorithm 2.3.7 $Tr(g^n)$ is computed by using the formulas (v) en (vii) from lemma 2.3.3. In this section we try to use formula (iv) instead. This is the formula that says that

$$c_{u+v} = c_u c_v - c_v^p c_{u-v} + c_{u-2v} \text{ for } u, v \in \mathbb{Z}.$$

This means that knowing c_u, c_v, c_{u-v} and c_{u-2v}, c_{u+v} can be computed. Suppose given $n \in \mathbb{N}$ we want to compute c_n. The way we are going to do this is make a so called *addition chain.*

Definition 3.1.1.
An addition chain $\mathcal{C}$ *of length k is a increasing sequence of natural numbers $< a_0, a_1, ..., a_k >$ such that for every $a_l \in \mathcal{C}$ except a_0 and a_1, there exists i, j with $0 \leq i \leq j < l$ such that $a_l = a_i + a_j$.*

Suppose we have such a chain with $n = a_k$ then we know that $c_n = c_{a_i + a_j}$ for some $i, j < k$ and $c_{a_i} = c_{a_g + a_h}$ for some $g, h < i$ and so on. So if we start with c_{a_0} and c_{a_1} we could use them to compute c_n by applying at most $k - 1$ times formula 2.3.3. But to do that we need for example $c_{a_i - a_j}$. Furthermore we want to start with c_0 and c_1, because they are known. That is why we define a special kind of addition chain.

Definition 3.1.2.
An addition chain $\mathcal{C}$ is called an XTR-chain *if for every $x \in \mathcal{C}$ (except 0 and 1) there exists $u, v \in \mathcal{C}$ such that $x = u + v$ and $|u - v|$ and $|u - 2v|$ are also $\in \mathcal{C}$. Furthermore 0 and 1 must be in $\mathcal{C}$.*

Example 3.1.3.
An example of such a chain is <0,1,2,3,5,8,13,21,34,47>. To compute c_{47} you do the following:
$2 = 1 + 1$ so $c_2 = c_1 c_1 - c_1^p c_0 + c_1^p$
$3 = 2 + 1$ so $c_3 = c_2 c_1 - c_1^p c_1 + c_0$
$5 = 3 + 2$ so $c_5 = c_3 c_2 - c_2^p c_1 + c_1^p$
$8 = 5 + 3$ so $c_8 = c_5 c_3 - c_3^p c_2 + c_1^p$
$13 = 8 + 5$ so $c_{13} = c_8 c_5 - c_5^p c_3 + c_2^p$
$21 = 13 + 8$ so $c_{21} = c_{13} c_8 - c_8^p c_5 + c_3^p$
$34 = 21 + 13$ so $c_{34} = c_{21} c_{13} - c_{13}^p c_8 + c_5^p$
$47 = 34 + 13$ so $c_{47} = c_{34} c_{13} - c_{13}^p c_{21} + c_8$

In this example the fact is being used that $c_{-n} = c_n^p$. Since this requires no multiplications in $GF(p)$ you don't have to make a difference between n and $-n$. That's why there are absolute values in definition 3.1.2.

Given an $n \in \mathbb{N}$ and an XTR-chain $\mathcal{C}$ with n as the largest element of $\mathcal{C}$ it takes $4 \cdot \#\mathcal{C}$ multiplications in $GF(p)$ to compute c_n, because every step takes four multiplications. Sometimes it takes even fewer multiplications because if $u = v$ computing c_{u+v} only takes two multiplications in $GF(p)$ (lemma 2.3.3(v)).

3.2 Fibonacci

To find the most efficient way to compute c_n with this method, we need to find the shortest XTR-chain in which n is contained. For Fibonacci numbers it is not hard to find a good XTR-chain. Because we often use the golden ratio let $G = \frac{1+\sqrt{5}}{2}$.

Lemma 3.2.1.
An XTR-chain for the k-th Fibonaccinumber f_k is $< f_0, f_1, ..., f_k >$ with $f_0 = 0$ and $f_1 = 1$.

Proof.
For $k = 1$ and $k = 2$ it is clear. Suppose the lemma is true for k with $k \geq 2$. Let $\mathcal{C} =< f_0, f_1, ..., f_{k+1} >$. For f_0 upto f_k it is true that they can be written as $u + v$ that fulfill the conditions of definition 3.1.2 because $< f_0, f_1, ..., f_k >$ is an XTR-chain.
Now $f_{k+1} = f_k + f_{k-1}$, so take $u = f_k$ and $v = f_{k-1}$ then u and v are elements of $\mathcal{C}$. Now $u - v = f_k - f_{k-1} = f_{k-1} + f_{k-2} - f_{k-1} = f_{k-2}$, so $|u - v| \in \mathcal{C}$. Furthermore $|u - 2v| = |f_k - 2f_{k-1}| = |f_{k-1} + f_{k-2} - 2f_{k-1}| = |f_{k-2} - f_{k-1}| = f_{k-1} - f_{k-2} = f_{k-3}$, so $|u - 2v|$ is also an element of $\mathcal{C}$. So $\mathcal{C}$ is an XTR-chain. With induction the lemma follows. □

Lemma 3.2.2. *Computing c_{f_n} by using this XTR-chain costs $4(n-3)$ multiplications in $GF(p)$ and is therefore faster then using the algorithm from chapter 2.*

Proof.
$c_{f_0} = c_0$, $c_{f_1} = c_1$ and $c_{f_2} = c_1$ so they are all given. Suppose $n = 3$ then c_{f_n} can be computed using 4 multiplications in $GF(p)$ with formula 2.3.3.
Suppose for n it is given that computing c_{f_n} costs $4(n-2)$ multiplications. Then it costs $4(n-2)$ multiplications to compute $c_{f_0}, ..., c_{f_n}$ and because $< f_0, ...f_n, f_{n+1} >$ is an XTR-chain for f_{n+1}, computing $c_{f_{n+1}}$ out of $c_{f_0}, ..., c_{f_n}$ costs 4 multiplications. This means that computing $c_{f_{n+1}}$ costs $4(n-2)+4 = 4(n-1) = 4((n+1)-2)$.
With induction the first part of the lemma follows.
Let $m = f_n$. Then $m = [\frac{1}{\sqrt{5}}G^n]$. Computing c_m with an XTR-chain costs $4(n-2)$ multiplications and with the method of chapter 2 it takes $8 \cdot {}^2\log(m)$ multiplications.
$m = [\frac{1}{\sqrt{5}}G^n] > \frac{1}{\sqrt{5}}G^n - 1$. Then $\sqrt{5}(m+1) > G^n$, so ${}^G\log(\sqrt{5}(m+1)) > n$.
This means that $4(n-2) < 4({}^G\log(\sqrt{5}(m+1)) - 2) = {}^G\log(m+1) + {}^G\log(\sqrt{5}) - 2 < {}^G\log(m+1) = 4\frac{{}^2\log(m+1)}{{}^2\log(G)} < 6\,{}^2\log(m+1) = {}^2\log((m+1)^6) < {}^2\log m^8 = 8\,{}^2\log m$ (if $m \geq 3$).
For $m < 3$ the lemma follows from direct computations. □

So for Fibonacci numbers we found a faster method, but is this the fastest way? This is not always the case. Look for example at $< 0, 1, 2, 4, 8 >$. This is an XTR-chain for the 6-th Fibonaccinumber 8. Using this chain it takes $4 \cdot 3 = 12$ multiplications to compute c_8 and that is faster then the $4(6-2) = 16$ multiplications from lemma 3.2.2.

3.3 Factorization

What we would like to know is how to find the shortest XTR-chain for a number n. With a Magmaprogram we computed the shortest XTR-chain for the numbers 1 to 1000. This is done by just trying all possible $u's$. Given n and u you can write down the XTR-chain with n and u in it by putting v $(= n - u)$, $|u - v|$ and $|u - 2v|$ in it. Then you write u as $v + (u - v)$ and again you put $|2v - u|$ and $|3v - 2u|$ in it and so on. When 1 is an element of the chain you stop. When 1 is not an element of the chain but there are no elements left to put in, you complete the chain by appending the shortest chain for the smallest number in the chain. How it works is shown in the following example.

Example 3.3.1.
If you want to compute an XTR-chain for $n = 44$ and $u = 28$ then you do the following:
You start with $< 44, 28 >$.
$44 - 28 = 16$, $28 - 16 = 12$ and $|28 - 32| = 4$ so you put them in: $< 44, 28, 16, 12, 4 >$.
Now you write $28 = 16 + 12$ and the elements you need to put in are: $16 - 12 = 4$ and $|16 - 24| = 8$ so you get: $< 44, 28, 16, 12, 8, 4 >$.
Now $16 = 8 + 8$, $8 - 8 = 0$ and $|8 - 16| = 8$. This gives $< 44, 28, 16, 12, 8, 4, 0 >$.
$12 = 8 + 4$, this gives no new elements.
$8 = 4 + 4$, this also doesn't give new elements.
So now you append the shortest XTR-chain for 4, which is $< 4, 2, 1, 0 >$. Then your chain is an XTR-chain: $< 44, 28, 16, 12, 8, 4, 2, 1, 0 >$

In appendix A you can find the table with for the numbers 1 to 1000 the length of the shortest chain and the u that gives the shortest chain (sometimes there are more). Furthermore you find the ${}^{G}\log(n)$, which we'll need later on.
In the previous section we saw that 8 has a shorter XTR-chain then the Fibonaccichain. This is due to the fact that 8 is divided by 2 so you can write it as $u + v$ with $u = v = 4$. When you do this you don't need other elements then 4 and 0 to apply formula 2.3.3. $|u - v|$ and $|u - 2v|$ are immediately in the chain. This leads to another way to make XTR-chains when given a way to make XTR-chains for primenumbers:

Theorem 3.3.2.
Given an XTR-chain $\mathcal{C}_1 = < n, n_1, n_2, ..., n_k >$ for an $n \in \mathbb{N}$ and an XTR-chain $\mathcal{C}_2 = < p, p_1, p_2, ..., p_l >$ for a primenumber p, you can make an XTR-chain for pn of length $k + l$.

Proof.
Every XTR-chain ends with 1,0 so $n_{k-1} = 1$ and $n_k = 0$. Now make the

following chain for pn: $\mathcal{C} = < pn, pn_1, pn_2, ..., pn_{k-1}, p_1, p_2, ..., p_l >$. Suppose $x \in \mathcal{C}$ then there are three possibilities: $x = pn_i$ for an $1 \leq i \leq k-2$, $x = pn_{k-1}$ or $x = p_i$ for an $1 \leq i \leq l$. In the first situation $n_i \in \mathcal{C}_1$ so there exist $u, v \in \mathcal{C}_1$ such that $n_i = u+v, |u-v| \in \mathcal{C}_1$ and $|u-2v| \in \mathcal{C}_1$. Then $pu, pv, |pu-pv| = p|u-v|$ and $|pu - 2pv| = p|u - 2v| \in \mathcal{C}$ and $x = pu + pv$. If $x = pn_{k-1}$ then $x = p$ so $x \in \mathcal{C}_2$. In the third situation x is also an element of $\mathcal{C}_2$. Elements of $\mathcal{C}_2$ fulfill the requirements of an XTR-chain in $\mathcal{C}_2$ and since $\mathcal{C}_2 \subseteq \mathcal{C}$, $\mathcal{C}$ is also an XTR-chain. □

This theorem means that given the shortest XTR-chains for primenumbers there's a way to make XTR-chains for all numbers by factoring and applying the theorem.
For the numbers 1 to 1000 you can verify that this method really gives the shortest XTR-chain. This leads to the question wether this is always the case, but this isn't the fact. In fact you find easily quit some counterexamples. Some of them can be found here. These are all examples of the form $n = pq$ in which p, q are primenumbers lesser then 1000.

Counterexample 3.3.3.
These examples are of the form $n = pq$ in which p, q are primenumbers lesser then 1000. First the shortest XTR-chain possible is given and then the XTR-chain found by applying theorem 3.3.2 and the shortest XTR-chains of p and q.

p:	23
q:	53
n:	1219
	$< 1219, 756, 463, 293, 170, 123, 76, 47, 29, 18, 11, 7, 4, 3, 2, 1, 0 >$
	$< 1219, 690, 529, 368, 207, 161, 115, 69, 46, 23, 13, 10, 7, 4, 3, 2, 1, 0 >$
p:	3
q:	421
n:	1263
	$< 1263, 781, 482, 299, 183, 116, 67, 49, 31, 18, 13, 8, 5, 3, 2, 1, 0 >$
	$< 1263, 693, 570, 447, 324, 201, 123, 78, 45, 33, 21, 12, 9, 6, 3, 2, 1, 0 >$

3.4 Continued fractions

Definition 3.4.1.
With every sequence $a_0, a_1, ..., a_k$ with $a_0 \in \mathbb{N}$ and $a_i \in \mathbb{N}^*$ for $1 \leq i \leq k$ we define the continued fraction $[a_0, a_1, ..., a_k]$ of this sequence inductively:

$$[a_0] = a_0$$

$$[a_0, a_1, ..., a_k] = a_0 + \frac{1}{[a_1, ..., a_k]}$$

This means that for example

$$\begin{aligned} [a_0, a_1, a_2, a_3] &= a_0 + \frac{1}{[a_1, a_2, a_3]} \\ &= a_0 + \frac{1}{a_1 + \frac{1}{[a_2, a_3]}} \\ &= a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3}}}. \end{aligned}$$

Every rational number can be written as such a continued fraction by applying the algorithm of Euclides. In the next example it is shown how this works.

Example 3.4.2.

$$\begin{aligned}\frac{65}{23} &= 2+\frac{19}{23}\\ \frac{23}{19} &= 1+\frac{4}{19}\\ \frac{19}{4} &= 4+\frac{3}{4}\\ \frac{4}{3} &= 1+\frac{1}{3}\end{aligned}$$

This means that

$$\begin{aligned}\frac{65}{23} &= 2+\frac{1}{\frac{23}{19}}\\ &= 2+\frac{1}{1+\frac{1}{\frac{19}{4}}}\\ &= 2+\frac{1}{1+\frac{1}{4+\frac{1}{\frac{4}{3}}}}\\ &= 2+\frac{1}{1+\frac{1}{4+\frac{1}{1+\frac{1}{3}}}}.\end{aligned}$$

which leads to the continued fraction $[2,1,4,1,3]$.

The connection between those continued fractions and the addition chains in the previous section can be made as follows.

Definition 3.4.3.
Given a continued fraction $[a_0,a_1,...,a_k]$ we define a sequence $p_{-1},p_0,...,p_k$ inductively:

$p_{-1} := 1$
$p_0 := a_k$
$p_i := a_{k-i}\cdot p_{i-1}+p_{i-2}$

This sequence can now be extended to an XTR-chain by following the next algorithm:

Algorithm 3.4.4.

1. *Remove p_{-1} from the sequence and reverse the sequence to $< p_k,p_{k-1},...,p_0 >$.*
2. *For all i with $1 \le i \le k$ do the following:*
   If $a_{k-i} > 1$ then append between p_i and p_{i-1} the numbers $p_{i-1}+p_{i-2}$, $2p_{i-1}+p_{i-2}$,...,$(a_i-1)\cdot p_{i-1}+p_{i-2}$.
3. *Compute an XTR-chain $< p_0,q_1,...,q_l >$ of p_0 and append this chain to the sequence.*

The XTR-chain you then get is $< p_k, (a_k-1)p_{k-1}+p_{k-2}, ..., p_{k-1}+p_{k-2}, p_{k-1}, ..., p_1, (a_{k-1}-1)p_0 + p_{-1}, (a_{k-1}-2)p_0 + p_{-1}, ..., p_0, q_1, ..., q_l >$.

Before the proof is given that this is truly an XTR-chain, an example will be given to show how the algorithm works.

Example 3.4.5.
Take for example the continued fraction from example 3.4.2 $[2, 1, 4, 1, 3]$. The "p-sequence" that belongs with this continued fraction is $[1, 3, 4, 19, 23, 65]$.
Following algorithm 3.4.4 we come to the following XTR-chain for 65:
$< 65, 42, 23, 19, 15, 11, 7, 4, 3, 2, 1 >$.

Theorem 3.4.6.
The chain you get by starting with a continued fraction of $\frac{n}{m}$ *with* $gcd(n, m) = 1$ *is an XTR-chain for* n.

Proof.
Suppose $\mathcal{C}$ is the chain we got after applying the algorithm. Let $w \in \mathcal{C}$, then there exists an $i \in \{1, ...k\}$ and a $j \in \mathbb{N}$ such that $w = jp_i + p_{i-1}$ with $1 \leq j \leq a_{k-i}$ or $w = q_i$ with $i \in \{1, ...l\}$ or $w = p_0$.
In the last two cases w is also an element of the XTR-chain $< p_0, q_1, ..., q_l >$, so there exists u, v in that chain and therefore in $\mathcal{C}$, such that $w = u+v$, $u, v, |u-v|$ and $|u - 2v| \in \mathcal{C}$.
In the other case $w = jp_i + p_{i-1}$ with $1 \leq j \leq a_{k-i+1}$, then take $u = (j-1)p_i + p_{i-1}$ and $v = p_i$. Then $w = u + v$ and u and v are both in $\mathcal{C}$.
$|u - v| = |(j-2)p_i + p_{i-1}|$.
If $j \geq 2$ then $|u - v| \in \mathcal{C}$. Else $|u - v| = | - p_i + p_{i-1}| = p_i - p_{i-1} = (a_{k-i} - 1)p_{i-1} + p_{i-2} \in \mathcal{C}$. Thus $|u - v| \in \mathcal{C}$.
$|u - 2v| = |(j-3)p_i + p_{i-1}|$.
If $j \geq 3$ then $|u - 2v| \in \mathcal{C}$.
If $j = 2|u - 2v| = | - p_i + p_{i-1}| \in \mathcal{C}$ (see above).
So if $j \geq 2$ then there exist u and $v \in \mathcal{C}$ such that $w = u + v$ and $|u - v|$ and $|u - 2v| \in \mathcal{C}$.
Suppose $j = 1$, then $w = p_i + p_{i-1}$. Now take $u = p_i$ and $v = p_{i-1}$. Then u and v are elements of $\mathcal{C}$ and $|u - v| = p_i - p_{i-1} = (a_{k-i} - 1)p_{i-1} + p_{i-2} \in \mathcal{C}$.
$|u - 2v| = |p_i - 2p_{i-1}|$. If $a_{k-i} \geq 2$, then $|u - 2v| = (a_{k-i} - 2)p_{i-1} + p_{i-2} \in \mathcal{C}$, else $a_{k-i} = 1$, so $|u - 2v| = |p_i - 2p_{i-1}| = | - p_{i-1} + p_{i-2}| = p_{i-1} - p_{i-2} = (a_{k-i+1} - 1)p_{i-2} + p_{i-3} \in \mathcal{C}$
So $\mathcal{C}$ fulfills the conditions of definition 3.1.2 and is therefore an XTR-chain. □

With this algorithm we found an improved way to search for short XTR-chains. Suppose you want to find a short XTR-chain for a natural number n then you can compute for all possible u's the continued fraction of n/u. Then you can apply algorithm 3.4.4 and you find the XTR-chain for n that contains u. This is a more efficient way than the method described in section 3.3, because computing continued fractions is already implemented in magma efficiently.

3.5 The golden section

In section 3.1 we saw that for Fibonaccinumbers there are easy to find XTR-chains that lead to an efficient computation of c_n. For a Fibonaccinumber f_n

the u that is used to make the XTR-chain is f_{n-1} which is $\frac{1}{G}f_n$. This leads to the thought that for arbitrary n we should also search this u near $\frac{1}{G}n$. To see if this is a good assumption we computed the shortest XTR-chain(s) with the continued fraction method for the numbers 1 to 10.000. A plot (see figure 1) was made in which you can find the values for u that lead to a shortest XTR-chain for every n.

PLAATJE MOET NOG KOMEN

In this plot you can see that there always lies an u that leads to a shortest chain close to $\frac{1}{G}n$. Because it is impossible to check all possible XTR-chains, we now search for a reasonable border to search within. The idea is that you compute the XTR-chains for n and u with u in the interval $(\frac{1}{G}n - p, \frac{1}{G}n + p)$ for a parameter p.

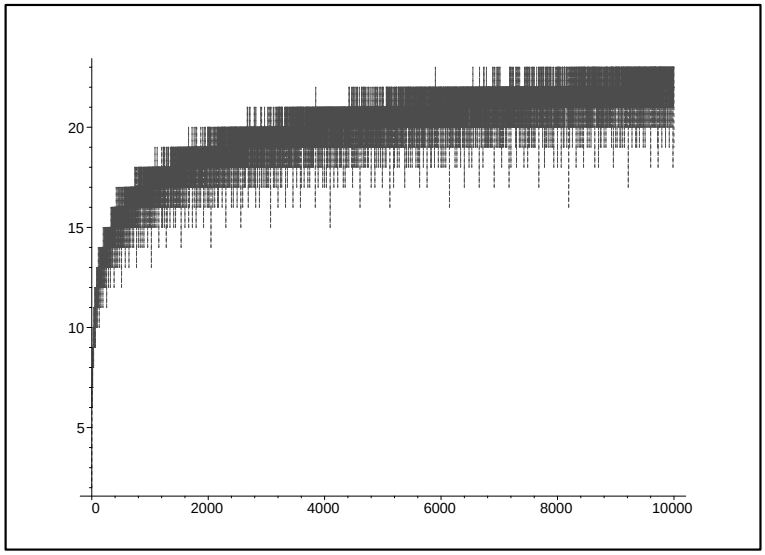
HIER KOMT DE TABEL MET VOOR VERSCHILLENDE PARAMETERS P EN GETALLEN VAN VERSCHILLENDE BITLENGTES DE LENGTE VAN HET KORTST GEVONDEN INTERVAL, DE REKENTIJD, HET AANTAL BEREKENINGEN IN GF(P) OP DEZE MANIER EN HET AANTAL BEREKENINGEN IN GF(P) OP DE OUDE MANIER (ZOALS BESCHREVEN IN HET ARTIKEL).

Depending on how much time the user has, he can choose a size of the parameter. In the table you can see that choosing $p = 6$ saves quit some computations in $GF(p)$ in not too much time. So for this value of p we tested some more. The results can be found in table 2.

HIER KOMT DE TABEL MET VOOR VERSCHILLENDE BITLENGTES DE GEMIDDELDE LENGTE EN TIJD MET DE STANDAARDAFWIJKING

Now we would like to now wether it is possible to predict the length of the shortest XTR-chain. That is why also a plot was made in which you can find the lengths of the shortest XTR-chains for the numbers 1 to 10.000 (see figure 2). You can see that they seem to be bounded by the line ${}^{G}\log(n)$, which again leads to the thought that finding a short chain has something to do with the Fibonacci numbers.

Figure 2: Length of the shortest XTR-chain for 1 to 10,000.



If it is really true that this gives an upper bound is something that still should be proved. If it is true computing c_n would cost at most $4 \cdot ({}^{G}\log(n) - 3)$ computations in $GF(p)$ instead of the $8 \cdot {}^{2}\log(n)$ computations necessary with the method from chapter 2. This can be proven in the exact same manner as lemma 3.2.2. Of course the statement is only useful if you can find that shortest XTR-chain in a reasonable amount of time.

References

[1] Arjen K. Lenstra, Eric R. Verheul, *The XTR public key system*, Advances in cryptography, Springer-Verlag, 2000

[2] Neal Koblitz, *Algebraic Aspects of Cryptography*, Springer Verlag, Berlin Heidelberg New-York, 1997

[3] Dominic Welsh, *Codes and Cryptography*, Oxford University Press, Oxford, 1988