DENNIS T. S. CUFLAT

COPROCESSADOR CRIPTOGRÁFICO PARA TRANSAÇÕES SEGURAS EM DISPOSITIVOS MÓVEIS

São Paulo
2015

DENNIS T. S. CUFLAT

COPROCESSADOR CRIPTOGRÁFICO PARA TRANSAÇÕES SEGURAS EM DISPOSITIVOS MÓVEIS

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a conclusão do curso de graduação em Engenharia de Computação, junto ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

Área de Concentração:

Engenharia de Computação

Orientador:

Wilson Vicente Ruggiero

Co-orientador:

Jonatas Faria Rossetti

São Paulo
2015

FICHA CATALOGRÁFICA

S. Cuflat, Dennis Tritapepe

Coprocessador Criptográfico para Transações Seguras em Dispositivos Móveis/ D. T. S. Cuflat. São Paulo, 2015.

96 p.

Monografia (Graduação em Engenharia de Computação) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

1. Segurança e criptografia com curvas elípticas #1. 2. Síntese e implementação de projeto de hardware em FPGA #2. 3. Software e interface de comunicação entre dispositivos #3. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais (PCS). II. t.

RESUMO

Este relatório documenta as motivações, especificação e o desenvolvimento de um dispositivo de hardware capaz de se conectar a uma plataforma host (por exemplo, um computador ou um dispositivo móvel) e realizar operações criptográficas de modo a prover uma camada adicional de segurança em transações online. O projeto é baseado nas demandas do cenário atual, onde cada vez mais as pessoas fazem uso de seus dispositivos móveis para realizar funções que lidam com dados que devem estar protegidos das vulnerabilidades da plataforma.

Propomos então um sistema baseado na criptografia de curvas elípticas, que permite gerar e validar assinaturas digitais. Como essa forma de criptografia é baseada na aritmética de corpos finitos, com a qual os processadores mais comuns não estão otimizados para trabalhar, projetaremos um coprocessador dedicado e capaz de realizar as funções passadas pela máquina host.

Ao longo do relatório, tratamos das bases teóricas de criptografia, corpos finitos e curvas elípticas, estabelecemos um cenário de uso, explicamos a metodologia de trabalho e recursos necessários para o desenvolvimento do projeto, fazemos a especificação de seus componentes de hardware e software, e, enfim, detalhamos os resultados obtidos a fim de verificar o sucesso do projeto.

ABSTRACT

This document provides the background, specification, and development of a hardware device that can be connected to a host platform (i.e a personal computer or a mobile device) and execute cryptographic operations, in order to provide an additional security layer for online transactions. This project is based on the demands of our current society, in which an increasing number of people make use of their mobile devices to do tasks that deal with sensitive data, which must be protected against the platform vulnerabilities.

Therefore, we present a system based on Elliptic Curves Cryptography, that can generate and validate digital signatures. Since this kind of cryptography is based on finite field arithmetic, which our usual processors are not optimized to deal with, we will design a dedicated coprocessor. It will be able to execute the cryptographic operations needed by the host machine.

Throughout this document, we will explain the theoretical foundation of cryptography, finite field, and elliptic curves; establish an use case scenario; explain the work methodology and resources needed for the project development; specify the software and hardware components and their synthesis; and, finally, test e report the results in order to verify if the project succeeded in its objectives.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

1 INTRODUÇÃO

1.1 Apresentação

Ao longo da formação no curso de Engenharia de Computação, somos expostos a uma vasta quantidade de matérias de diferentes áreas de atuação, sejam projetos de hardware, software, redes e segurança, passando pelos conceitos físicos e matemáticos com os quais formamos a nossa base nos primeiros anos de curso. Sendo assim, este trabalho de conclusão de curso apresenta, em princípio, a tarefa de tentar agregar o conhecimento adquirido em um projeto coeso com uma aplicação relevante ao nosso cenário tecnológico atual.

Na idealização do tema para este projeto, foi definido que, de modo a melhor atender a essa proposta, o foco principal deveria ser a síntese de um hardware, dando continuidade aos estudos de organização e arquitetura de computadores. Adicionalmente, um software companheiro seria desenvolvido como forma de interface entre o usuário e o dispositivo, rodando em uma máquina host ao qual este dispositivo se conecta.

Dentre as possibilidades de aplicação, que também incluíam processadores gráficos, gerenciadores de recursos físicos de uma residência, e uma rede de sensores sem fio, chegamos a um tema que, não apenas abrange uma grande parte de áreas da computação, como também é uma demanda constante no mercado, e que permite adquirir um conhecimento novo e significante no meio profissional: segurança de dados. Quando tratamos de segurança de dados, consideramos desde os meios físicos pelos

quais passa uma informação e as camadas de serviço utilizadas para transmiti-la, até o próprio formato no qual ela está codificada. Com o avanço crescente da tecnologia, especialmente com o advento dos dispositivos móveis, o leque de possibilidades que temos em nossas mãos para nos comunicar aumentou, e com ele, também a quantidade de vulnerabilidades de segurança.

É uma consequência natural que, neste novo cenário, surjam constantemente novas demandas para a maior comodidade do usuário: não basta mais apenas ser capaz de se comunicar com outros dispositivos, o dispositivo móvel também deve ser um substituto viável para realizar operações antes desempenhadas por outros aparelhos eletrônicos, ou restritas ao meio não digital, e que podem envolver dados mais delicados, como, por exemplo, transações financeiras. Sistemas como o «Apple Pay» (compatível com o iPhone), que permite realizar pagamentos através de NFC e validação por sensores biométricos, já são uma realidade. E sistemas como esse também trazem consigo uma demanda ainda maior de segurança de informação.

Uma das mais importantes camadas de segurança de dados é a criptografia, um dos temas de estudo do curso de redes. Formas mais robustas de criptografia, no entanto, demandam mais recursos computacionais, visto que normalmente fazem uso de ferramentas matemáticas com as quais os dispositivos de hardware que utilizamos diariamente não foram projetados para trabalhar eficientemente.

A técnica de criptografia de curvas elípticas é especialmente interessante para essa abordagem. É uma técnica poderosa, cuja segurança está baseada no problema dos logaritmos discretos e, portanto, é computacionalmente inviável obtermos a mensagem original a partir da mensagem cifrada sem o conhecimento da chave privada associada à chave pública utilizada na etapa de cifração (HEKERSON; MENEZES; VANSTONE, 2004). Essa técnica de criptografia faz uso de aritmética de corpos finitos, e é nesse ponto que a maior parte dos hardware comerciais apresenta um gargalo de eficiência.

Dessa forma, a possibilidade de um dispositivo de hardware otimizado para traba-

lhar com este tipo de aritmética, e que possa ser conectado a uma máquina host para realizar funções criptográficas para uso em uma aplicação local, se torna bastante atraente, adicionando uma camada robusta de segurança para os sistemas que utilizamos. Fazemos deste, então, o tema para o projeto de conclusão de curso.

1.2 Motivação

Estabelecemos a hipótese de um cenário para este projeto: a compra de um item através de uma loja virtual. Com ele, podemos identificar um problema que implique em uma demanda, e uma possível solução que justifica um produto. Ao final desta descrição, podemos então especificar os objetivos exatos desse projeto. Ao realizarmos a compra em uma loja virtual, estamos fazendo uso de uma criptografia assimétrica, baseada em um par de chaves: temos acesso à chave pública da loja e, assim, ciframos alguns dados importantes de pagamento durante a transação que somente poderão ser revelados para a loja, através do uso da chave privada que apenas ela conhece e que está associada à chave pública utilizada.

Identificamos duas questões pertinentes: primeiramente, uma vez que estamos cada vez mais utilizando dispositivos móveis, expostos a vários novos canais de comunicação e brechas de segurança, para realizar compras e transações, nossos dados tornam-se mais vulneráveis a alguns ataques que buscam contornar a segurança oferecida pela criptografia através da análise de vazamento de dados, como consumo de energia e sinais eletromagnéticos. Dessa forma, é importante que utilizemos métodos mais robustos de criptografia, que permitam obter um nível maior de segurança, mas que ao mesmo tempo exijam um maior desempenho computacional.

Adicionalmente, o tipo de criptografia assimétrica que utilizamos traz a desvantagem de assegurar a identidade apenas da entidade portadora de chave privada, ou seja, a loja. Em um ambiente onde estamos cada vez mais expostos a fraudes, torna-se

cada vez mais necessário que possamos assegurar que uma transação financeira feita em nome de uma pessoa através de seu dispositivo tenha sido, de fato, feita por essa pessoa.

Nesse contexto, a posse de uma chave privada permite gerar uma assinatura digital, sujeita a validação que comprove a origem dos dados, tornando mais difícil a realização de uma transação fraudulenta. Propomos, portanto, um sistema onde:

- Tanto usuários como comércio tenham um par de chaves pública e privada.
- As assinaturas digitais sejam geradas por um método de criptografia mais robusto.

Obviamente, tanto o aumento no número de processos de criptografia quanto sua qualidade vão demandar recursos computacionais específicos. Nesse cenário, justificamos o desenvolvimento de um coprocessador criptográfico baseado em curvas elípticas para se adequar a esse sistema, que oferece uma alternativa para as demandas atuais de segurança.

1.3 Objetivos

O objetivo deste projeto de conclusão de curso é a síntese de um dispositivo de hardware que, conectado a uma máquina host, é capaz de realizar funções de criptografia e geração de assinatura digital fazendo uso de um coprocessador dedicado à aritmética de corpos finitos. Esse dispositivo visa adicionar uma camada de segurança para aplicações (móveis, principalmente), que trabalham com dados sigilosos.

Com este objetivo em mente, podemos listar os requerimentos necessários para que o projeto entregue um resultado adequado. Segue abaixo:

- Devemos estabelecer um cenário de uso para guiar o projeto do dispositivo. Focaremos em uma vertente baseada em validação de uma assinatura digital: para o

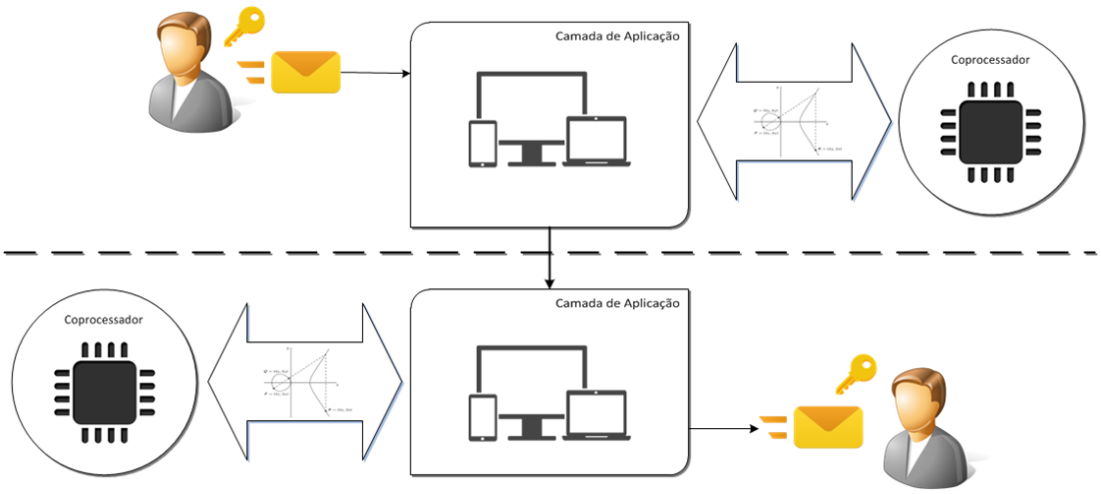


Figura 1: Plataformas trocam dados criptografados pelos dispositivos externos

consumidor, uma transação financeira móvel que requer a geração de uma única assinatura (foco em tempo de resposta); e para a loja, uma validação em lotes realizada em uma máquina dedicada (foco em vazão).

- Devemos sintetizar um dispositivo de hardware que atua como um coprocessador e consegue, de maneira eficiente, realizar cálculos de aritmética de corpos finitos, especialmente aqueles necessários para calcular pontos de curvas elípticas.

- O dispositivo deve ser capaz de receber uma mensagem original, parâmetros de uma curva elíptica, e uma chave pública, e devolver a mensagem criptografada.

- O dispositivo deve ser capaz de receber uma mensagem criptografada, parâmetros de uma curva elíptica, e uma chave privada, e devolver a mensagem original.

- O dispositivo deve ser capaz de gerar uma assinatura digital para quaisquer mensagens criptografadas, e ser capaz de validar uma assinatura digital.

- O dispositivo deve ser sintetizado a partir de uma descrição em VDHL, e implementado em uma FPGA.

- Devemos desenvolver um aplicativo a ser rodado na maquina host, que simule um caso de uso que requer uma camada adicional de segurança de dados. Este

aplicativo deve ser codificado em uma linguagem orientada a objetos, e deve ser capaz de se comunicar com o dispositivo de hardware.

- A máquina host, idealmente, deve ser uma plataforma móvel, como um smartphone com sistema operacional Android. Caso seja necessário simplificar uma das frentes do projeto por limitação de tempo, podemos utilizar um PC como máquina host.

- Todo o processo de uso do aplicativo e realização das tarefas de criptografia pelo dispositivo de hardware deve acontecer dentro de uma janela de tempo considerada aceitável para o caso de uso tratado, e, portanto, eficiência é um dos requisitos mais importantes.

- Além da eficiência computacional em termos de tempo, o projeto deve ser viável em relação a outras métricas de desempenho de hardware, como área e consumo de energia, de modo que seja possível inseri-lo na forma de um dispositivo embarcado.

1.4 Metodologia

Com os objetivos definidos, precisamos definir alguns critérios com os quais, ao final do projeto, julgaremos se este cumpre ou não com estes objetivos. Caso positivo, com que grau de eficiência, e, caso negativo, as razões pelas quais não conseguimos atingir os resultados esperados.

- O dispositivo faz o que é esperado?

  Precisamos, aqui, submeter o dispositivo ao caso de uso específico que será detalhado. Faremos uma análise de entradas, saídas esperadas, e resultados, para determinar seu funcionamento de acordo com os parâmetros pré-determinados.

- O dispositivo cumpre com os requisitos de tempo?

Nos nossos casos de uso, nos deparamos com duas situações específicas: para o consumidor, é importante que o dispositivo tenha um tempo de resposta pequeno, e para a loja, é importante que o dispositivo tenha vazão de dados grande. Devemos então definir o que consideramos tolerável, e se o tempo e vazão registrados pelo dispositivo se enquadram nessa categoria.

- O dispositivo é viável para aplicações móveis?

  Aqui, devemos nos preocupar com dois pontos importantíssimos se desejamos que este dispositivo possa funcionar como periférico ou como hardware embarcado de um sistema móvel: área e consumo de energia. Devemos analisar se o hardware resultante pode ser fisicamente integrado a um smartphone sem aumentar seu tamanho, e se ele não afeta de forma considerável seu consumo de bateria, um dos problemas mais frequentes enfrentados por essa plataforma.

- O dispositivo é necessário para o sistema?

  Talvez uma das questões mais importantes, ainda que fácil de ser ignorada: de nada adianta desenvolver este dispositivo se, no sistema que propomos, ele pode ser facilmente substituído por alternativas de software. Para verificar isso, vamos simular seu funcionamento em software e comparar nossos resultados como aqueles obtidos usando o dispositivo.

2 FUNDAMENTOS TEÓRICOS

2.1 Conceitos de Criptografia

Este projeto é baseado em um dispositivo capaz de realizar operações de criptografia assimétrica e assinatura digital. Dessa forma, é conveniente estabelecer os principais conceitos que utilizaremos no restante do projeto.

Fazemos uso de criptografia quando queremos estabelecer um canal seguro de comunicação. A criptografia assimétrica é baseada na geração de um par de chaves (pública e privada) através de um problema matemático considerado computacionalmente inviável de ser resolvido sem estes parâmetros, como o problema dos logaritmos discretos (HEKERSON; MENEZES; VANSTONE, 2004).

Tomemos uma função f(a,x) com estas características. Teremos, então, uma função simétrica g(b,y) de modo que, quando y=f(a,x), g(b,y)=x. Podemos aplicar estes conceitos a criptografia gerando um par de chaves [a,b] onde [a] é chave pública e [b] é chave privada. Detalhamos abaixo:

- Mapeamos uma string de dados na forma de um valor [x].
- Fazemos uso da função [f] e da chave pública [a] do destinatário, de modo a calcular uma string criptografada y=f(a,x).
- O destinatário da mensagem então faz uso da função [g] e da chave privada [b] para revelar a mensagem original x=g(b,y).

- É computacionalmente inviável calcular [x] a partir de [y], [a] e as funções [f,g], sem, no entanto, conhecer a chave privada [b].

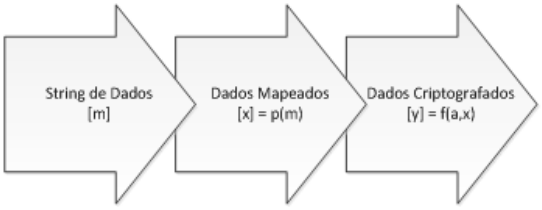


Figura 2: Etapas da Cifração

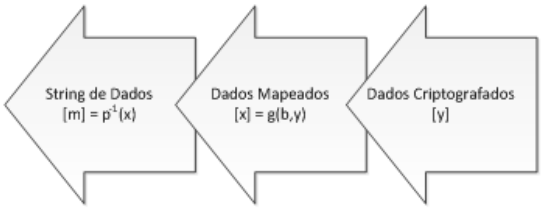


Figura 3: Etapas da Decifração

Os métodos de criptografia assimétrica também nos permitem gerar e validar assinaturas digitais, métodos para assegurar a identidade do remetente de uma mensagem e não violação de seu conteúdo. Isso é feito gerando uma assinatura que depende de uma função hash sobre seu conteúdo criptografado com a chave privada do remetente. Detalhamos abaixo:

- Fazemos uso de uma função de hash [h] válida, e escolhida a partir de parâmetros de segurança determinados, para gerar uma string [s] de tamanho adequado a partir da mensagem original.
- Mapeamos essa string de dados [s] na forma de um valor [y].
- Fazemos uso da função [g] e da chave privada [b] de modo a gerar uma assinatura digital [x]=g(b,y).
- Para validarmos a assinatura digital, tomamos a mensagem original e passamos pelo mesmo procedimento de hash e mapeamento até chegar ao valor de [y].

- Fazemos, então, uso da função [f] e chave pública [a] para calcular [y]=f(a,x).
- Pelo inverso da função de mapeamentos, resgatamos a string de hash [s].
- Caso os valores de [s] sejam iguais, então a assinatura digital foi corretamente validada. Caso os valores sejam diferentes, então as informações a respeito do remetente (na forma da chave [b]) ou do conteúdo da mensagem (na forma da string [s]) foram violados.
- Com a assinatura digital, 1) Nenhum terceiro pode passar uma mensagem em nome de um remetente sem sua chave privada [b]; 2) Nenhum terceiro pode violar o conteúdo de uma mensagem enviada [s] por um remetente sem que a alteração seja descoberta; 3) O remetente não pode negar ter sido, de fato, a origem de uma mensagem.

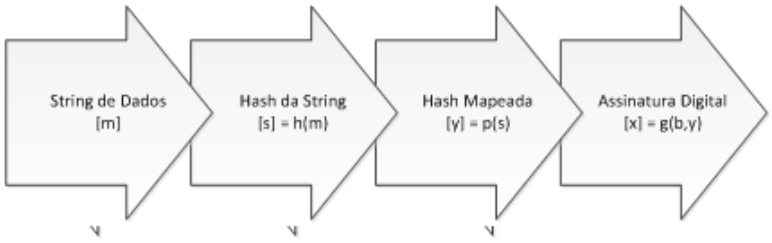


Figura 4: Etapas da geração de uma assinatura digital

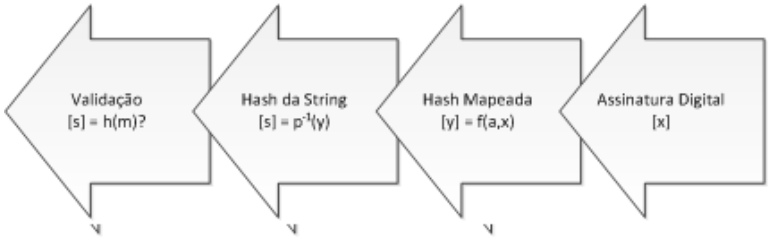


Figura 5: Etapas da validação de uma assinatura digital

2.2 Aritmética de Corpos Finitos

O tipo de criptografia que vamos utilizar neste projeto depende da aritmética dos corpos finitos, que servirá como base teórica de toda a especificação do coprocessador. Como as unidades aritméticas encontradas no hardware que utilizamos diariamente não estão otimizadas para trabalhar com esse tipo de aritmética, a eficiência computacional das operações de criptografia não é a ideal, motivo pelo qual esta implementação é importante para cumprir com os requisitos do sistema proposto. Explicamos os conceitos principais: um campo de corpos finitos é a abstração de um conjunto numérico abeliano, com número limitado de elementos e que satisfaz as seguintes propriedades (BROWN, 2009)

- Para dois elementos [a,b] quaisquer do conjunto, qualquer operação sobre estes elementos resulta em [c] que também está no conjunto.
- O conjunto aceita operação de adição, com identidade [0].
- O conjunto aceita operação de multiplicação, com identidade [1].
- Tanto operações de adição como multiplicação são associativas e comutativas.
- Para qualquer elemento [a] do conjunto, há um elemento inverso aditivo b no conjunto, tal que a+b=0.
- Para qualquer elemento [a] do conjunto, há um elemento inverso multiplicativo b no conjunto, tal que a*b=1.
- Operações de subtração e divisão são representadas como operações de soma e multiplicação dos elementos inversos.

Em um conjunto de corpos finitos, o tipo de aritmética utilizada é modular, definida sobre a ordem (número total de elementos) do conjunto, ou seja, para um conjunto

primo F = {0,1,2,3,4...n}, as operações são dadas em «módulo n». Tomamos como exemplo um conjunto de ordem 7 dado por F = {0,1,2,3,4,5,6}. As operações então serão dadas em «módulo 7». Exemplos:

- Soma: 5 + 4 = 9 mod 7 = 2;
- Multiplicação: 2 * 6 = 12 mod 7 = 5;
- Inversão Aditiva: -2 = 5, pois 2 + 5 = 7 mod 7 = 0;
- Inversão Multiplicativa: $\frac{1}{2} = 4$, pois 2 * 4 = 8 mod 7 = 1;

Existem dois tipos principais de corpos finitos: Aqueles definidos em um campo primo *FP*, que consiste em um conjunto de inteiros de ordem igual a um número primo, e que foi exemplificado nas operações acima; e aqueles definidos em um campo binário $F2^m$. Os corpos finitos binários têm propriedades particularmente convenientes em uma implementação de hardware (BROWN, 2009).

O campo finito $F2^m$ pode ser representado na forma de um polinômio:

$$F2^m = \{a_{m-1}x_{m-1} + a_{m-2}x_{m-2} + ... + a_1x + a_0\}$$

E os coeficientes am deste polinômio fazem parte do conjunto primo *F2*=0,1. Portanto, um conjunto de corpos finitos binários de grau m=4 $F2^4$ contém os seguintes elementos (polinômios): $F2^4 = \{[0], [1], [x], [x + 1], [x^2], [x^2 + 1], [x^2 + x], [x^2 + x + 1], [x^3], [x^3 + 1], [x^3 + x], [x^3 + x + 1], [x^3 + x^2], [x^3 + x^2 + 1], [x^3 + x^2 + x], [x^3 + x^2 + x + 1]\}$.

E, da mesma forma que em um conjunto de corpos primos, as operações são modulares sobre a ordem, em um conjunto de corpos binários, as operações são modulares sobre um polinômio irredutível (não fatorável) de grau m. Um polinômio $F2^4$ irredutível possível é $[x4 + x + 1]$. Exemplificamos abaixo as operações sobre um conjunto de corpos finitos binários:

- Soma: $[x^3 + x^2 + 1] + [x^2 + x + 1] = [x^3 + x]$;
- Inversão Aditiva: $-[x^3+x^2+1] = [x^3+x^2+1]$, pois $[x^3+x^2+1]+[x^3+x^2+1] = 0$, Para qualquer elemento [a] de $F2^m$, $[-a] = [a]$;
- Multiplicação: $[x^3 + x^2 + 1] * [x^2 + x + 1] = [x^5 + x + 1] mod [x^4 + x + 1] = [x^2 + 1]$;
- Inversão Multiplicativa: $1/[x^3 + x^2 + 1] = [x^2]$, pois $[x^2] * [x^3 + x^2 + 1] = [x^5 + x^4 + x^2] mod [x^4 + x + 1] = 1$;

É interessante notar que todos os corpos finitos binários podem ser expressos diretamente como uma sequência de bits, o que é ideal para implementação em hardware. Reescrevemos o exemplo acima nessa notação:

- Soma: [1101] + [0111] = [1010];
- Inversão Aditiva: -[1101] = [1101];
- Multiplicação: [1101] * [0111] = [0101];
- Inversão Multiplicativa: 1/[1101] = [0100];

2.3 Aritmética de Curvas Elípticas

Uma vez que já estabelecemos o conceito de corpos finitos, podemos apresentar o conceito das curvas elípticas sobre corpos finitos. Estas curvas são definidas pela Equação de Weierstrass, dada abaixo (HEKERSON; MENEZES; VANSTONE, 2004)

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

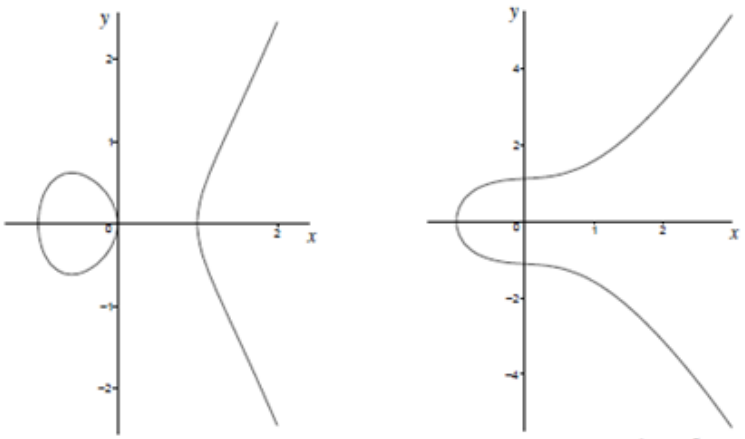
Os coeficientes [a] da equação precisam respeitar uma condição matemática que implica que o discriminante da curva deve ser diferente de zero (o que impede que

a curva tenha mais de uma reta tangente em qualquer ponto). Não entraremos em detalhes a respeito da natureza do discriminante pois a teoria não é o foco deste projeto, sendo suficiente apresentar a forma geral das curvas elípticas e garantir que os coeficientes com que trabalharemos seguem esta regra.

Se K é um campo de corpos finitos, então uma curva elíptica definida sobre este campo é:

$$E(K) = \{(x, y) \in KxK : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{O\}$$

Onde [O] é um ponto chamado «Ponto no Infinito», definido como a identidade [0] da curva elípitica.



(a) $E_1 : y^2 = x^3 - x$ (b) $E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

Figura 6: Exemplos de curvas elípticas (HEKERSON; MENEZES; VANSTONE, 2004)

Como estamos trabalhando com campos primos, de ordem P>2, podemos utilizar um recurso de mudança de variáveis para chegar em uma Equação de Weierstrass simplificada para este caso. Fazemos:

$$(x, y) => (\frac{x - 3{a_1}^2 - 12a_2}{36}, \frac{y - 3{a_1}^2x}{216} - \frac{{a_1}^3 + 4a_1a_2 - 12a_3}{24})$$

E chegamos à equação:

$$E : y^2 = x^3 + ax + b$$

Os pontos que satisfazem a estas condições em uma curva elíptica formam um grupo abeliano de ordem #E (número de pontos no grupo) junto ao ponto [O]. Portanto, também podemos aplicar operações de grupos sobre eles. Sejam $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ pontos de uma curva E, definimos P+Q, geometricamente, como sendo a reflexão sobre o eixo x do ponto de intersecção entre a reta PQ e um terceiro ponto da curva E. Quando P=Q, definimos 2P, geometricamente, como a reflexão sobre o eixo x do ponto de intersecção entre a reta tangente a P e um terceiro ponto da curva E.

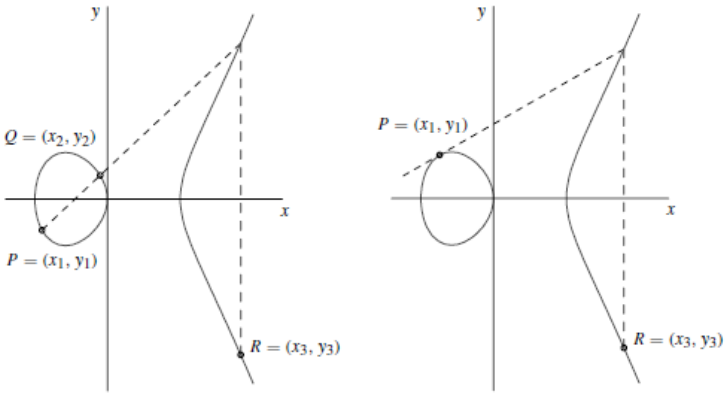


Figura 7: Representação gráfica das operações de soma sobre pontos de uma curva elíptica (HEKERSON; MENEZES; VANSTONE, 2004)

Respeitando as propriedades da identidade [0], sabemos que, para qualquer ponto P da curva E:

- $P + O = O + P = P$
- $P + (-P) = O$
- $O = -O$

Apesar das definições geométricas, podemos chegar a equações sobre corpos finitos para as operações de curvas elípticas. Temos abaixo as propriedades de grupos para os dois tipos simplificados de curvas sobre campos binários, sempre considerando $P(x_1, y_1)$ e $Q(x_2, y_2)$ como pontos da curva E (HEKERSON; MENEZES; VANSTONE, 2004)

Se $E : y^2 = x^3 + ax + b$

- Negativo: $R(x_3, y_3) = -P(x_1, y_1)$

  $x_3 = x_1$

  $y_3 = -y_1$

- Soma: $R(x_3, y_3) = P(x_1, y_1) + Q(x_2, y_2)$

  $x_3 = ((y_2 - y_1)/(x_2 - x_1))^2 - x_1 - x_2$;

  $y_3 = ((y_2 - y_1)/(x_2 - x_1))(x_1 - x_3) - y_1$;

- Dobro: $R(x_3, y_3) = 2 * P(x_1, y_1)$

  $x_3 = ((3{x_1}^2 + a)/(2y_1))^2 - 2x_1$;

  $y_3 = ((3{x_1}^2 + a)/(2y_1))(x_1 - x_3) - y_1$;

Vamos usar como exemplo um campo [K] do tipo [$F29$]. Uma curva elíptica do tipo [$E : y^2 = x^3 + ax + b$] é definida sobre este campo, com coeficientes [a=4] e [b=20]. Temos:

$$E : y^2 = x^3 + 4x + 20$$

A equação pode, então, ser resolvida com aritmética de corpos finitos sobre [$F29$], e podemos encontrar o conjunto de pontos P(x,y) que compõe o grupo abeliano:

- Se [x=0], [$y^2 = (0^3 + 4 * 0 + 20) = 20 mod 29$] => [y=7] ou [y=22];

- Se [x=1], [$y^2 = (1^3 + 4 * 1 + 20) = 25 mod 29$] => [y=5] ou [y=24];
- (...)
- Se [x=7], [$y^2 = (7^3 + 4 * 7 + 20) = 14 mod 29$] => [Não há y em *F29*];
- (...)
- Se [x=27], [$y^2 = (27^3 + 4 * 27 + 20) = 4 mod 29$] => [y=2] ou [y=27];

Conjunto de pontos de E: [(O), (0,7), (0,22), (1,5), (1,24), (2,6), (2,23), (3,1), (3,28), (4,10), (4,19), (5,7), (5,22), (6,12), (6,17), (8,10), (8,19), (10,4), (10,25), (13,6), (13,23), (14,6), (14,23), (15,2), (15,27), (16,2), (16,27), (17,10), (17,19), (19,13), (19,16), (20,3), (20,26), (24,7), (24,22), (27,2), (27,27)]
Ordem #E = 37
Exemplo de Negativo: -(5,22) = (5,7);
Exemplo de Soma: (5,22) + (16,27) = (13,6);
Exemplo de Dobro: 2*(5,22) = (14,6);

Observe que, apesar de estarmos utilizando campos de ordem relativamente baixa como exemplo, em aplicações reais as ordens são muito maiores, do tipo [*FP* , P » 2100]. Nestes casos, é possível mapear códigos inteiros na forma de uma coordenada [x] para o cálculo de um ponto [P] válido, sendo esta a forma mais comum de preparar uma mensagem para ser cifrada utilizando criptografia de curvas elípticas. Uma mensagem maior poderia ser fragmentada em mais de um ponto, sendo possível até mesmo mapear diretamente códigos ASCII a pontos de uma curva elíptica (BROWN, 2009).

Observamos também que nem todo elemento de um campo [*FP*] é uma coordenada [x] possível para um ponto [P]. Nestes casos, fazemos uso de um multiplicador [t] de modo que, para todo código [m] possível, há uma coordenada válida no intervalo [(t)m,(t+1)m].

2.4 Assinatura Digital com Curvas Elípticas

Os itens anteriores apresentam todos os conceitos teóricos necessários para que, agora, possamos implementar um método de assinatura digital fazendo uso de curvas elípticas. Antes disso, no entanto, aplicamos estes conceitos ao caso mais simples de cifrar uma mensagem qualquer [m]. Tomemos uma curva elíptica [E] definida sobre um campo [FP]. Dentre o conjunto de pontos que satisfazem a Equação de Weierstrass, tomamos um ponto [P] tal que [P] $\in$ [O] (Ponto no Infinito). Este ponto [P] formará um grupo abeliano cíclico sobre esta curva por multiplicação escalar ([P],[2P],[3P],[4P]...) de ordem [n] (quantidade de pontos deste grupo).

Sendo assim, podemos escolher um valor [d] no intervalo [1,n] de forma que obtemos um ponto [Q]=[dP]. Uma vez que temos o inteiro [d] e o ponto [P], é fácil obter [Q], mas se, no entanto, tivermos apenas [P] e [Q], o problema de encontrar o valor [d] é justamente aquele equivalente ao problema dos logaritmos discretos, sendo computacionalmente inviável. Podemos fazer uso dessa propriedade e manipular dados de modo a gerar uma operação criptográfica. Se escolhermos um outro inteiro [k] no intervalo [1,n], podemos fazer [C1] = [kP], e assim (HEKERSON; MENEZES; VANSTONE, 2004) :

$$d[C1] = d[kP] = k[dP] = [kQ]$$

Com isso, podemos cifrar uma mensagem [m] em um par [C1],[C2] fazendo:

- A mensagem [m] é mapeada a um ponto [M] da curva [E];
- Selecionamos um [k] no intervalo [1,n];
- Calculamos [C1] = [kP];
- Calculamos [C2] = M + [kQ];

E deciframos esta mesma mensagem fazendo:

- Calculamos [M] = [C2] - d[C1];
- Extraimos a mensagem [m] mapeada em [M];

Neste contexto, se temos um domínio com [F] e [E] bem definidos, juntamente a um ponto escolhido [P], podemos fazer uso do par [Q],[d] como um par de chaves pública e privada, respectivamente. Este é o método de criptografia de curvas elítpticas. Podemos expandir este conceito de criptografia para o caso de geração e validação de assinatura digital, que já conta com alguns protocolos bem conhecidos para o caso de curvas elípticas. Um esquema particularmente eficiente e conhecido é o ECDSA (Elliptic Curve Digital Signature Algorithm). Para fazer uso do esquema, a primeira coisa que devemos estabelecer é um domínio sobre o qual trabalharemos. Um domínio é, genericamente, dado pela forma (HEKERSON; MENEZES; VANSTONE, 2004) :

$$D = q, FR, S, a, b, P, n, h$$

Onde:

- <q> é a ordem do campo [F] de corpos finitos sobre o qual se define a curva elíptica [E];
- <FR> é a representação utilizada pelos elementos do campo [F] utilizado;
- <S> é a semente que gera os coeficientes aleatórios para uma curva elíptica;
- <a> e <b> são os coeficientes da curva elíptica;
- <P> é um ponto qualquer da curva [E] que define um grupo abeliano cíclico através de multiplicação escalar;

- <n> é a ordem do grupo definido por [P], ou seja, quantidade de pontos no grupo cíclico;
- <h> é chamado cofator, dado por #E(F)/n, e pode ser utilizado como parâmetro de segurança para alguns dos ataques mais comuns.

Adicionalmente, como especificado no item sobre protocolos de assinatura digital, precisamos definir uma função de hash [H] pela qual passaremos a mensagem [m] a ser assinada. Com estes parâmetros, definimos dois algoritmos: Um algoritmo que gera uma assinatura digital para uma mensagem [m] na forma de um par de inteiros [r,s] através de uma chave privada [d]; e um algoritmo que valida uma assinatura digital para uma mensagem [m] através de uma chave pública [Q]. Seguem os algoritmos abaixo (HEKERSON; MENEZES; VANSTONE, 2004) :

Geração de uma Assinatura Digital:

INPUT: Domínio [D], Chave Privada [d], mensagem [m], função de hash [H].

OUTPUT: Assinatura digital [r,s].

- Selecionamos um inteiro [k] no intervalo [1,n];
- Calculamos [kP] = (x_1, y_1) e convertemos $[x_1]$ para um inteiro $[X_1]$;
- Calculamos [r] = $[X_1]$ mod [n]. Se [r]=0, escolhemos um outro [k];
- Calculamos [e] = H(m);
- Calculamos [s] = [k]-1(e+dr) mod [n]. Se [s]=0, escolhemos um outro [k];
- A assinatura digital é dada na forma do par (r,s);

Validação de uma Assinatura Digital:

INPUT: Domínio [D], Chave Pública [Q], mensagem [m], função de hash [H], Assinatura digital [r,s].

OUTPUT: Validação da assinatura digital: [ACEITA] ou [REJEITA].

- Os valores no par [r,s] estão dentro do intervalo [1,n]? Se não, [REJEITA];
- Calculamos [e] = H(m);
- Calculamos [w] = [s]-1 mod [n];
- Calculamos [u_1] = [e][w] mod [n];
- Calculamos [u_2] = [r][w] mod [n];
- Calculamos [Z] = [$u_1 * P$] + [$u_2 * Q$];
- O ponto [Z] é o Ponto no Infinito [O]? Se sim, [REJEITA];
- Convertemos a coordenada [x_1] de Z(x_1, y_1) para um inteiro [X_1];
- Calculamos [v] = [X_1] mod [n];

v =[r]? Caso positivo, [ACEITA]; Caso negativo, [REJEITA];

São estes os algoritmos de assinatura digital que vamos utilizar neste projeto.

Exemplo de Aplicação: Abaixo, fazemos um exemplo simples com o grupo [*F29*] cujo grupo de pontos já foi determinado anteriormente. Ou seja, temos a curva:

$$E : y^2 = x^3 + 4x + 20$$

E assumimos P=(1,5). Se gerarmos um par de chaves com chave privada [d=7], temos como chave pública Q=(24,22). Adicionalmente, se uma mensagem [m] gera um hash H(m)=17, mapeada para M=(17,10), podemos seguir o algoritmo:

Geração de uma Assinatura Digital:

- Selecionamos um inteiro [k] no intervalo [1,n];

  Escolhemos [k]=15;

- Calculamos [kP] = (x_1, y_1) e convertemos $[x_1]$ para um inteiro $[X_1]$;

  *Calculamos 15*P = (3,1), X=3;*

- Calculamos [r] = $[X_1]$ mod [n]. Se [r]=0, escolhemos um outro [k];

  Calculamos [r] = 3 mod 37 = 3;

- Calculamos [e] = H(m);

  Função de Hash já aplicada, [e] = 17;

- Calculamos [s] = [k]-1(e+dr) mod [n]. Se [s]=0, escolhemos um outro [k];

  *Calculamos [s] = (17+7*3)/15 mod 37 = (38/15) mod 37 = 5;*

- A assinatura digital é dada na forma do par (r,s);

  A Assinatura Digital é o par (3,5).

Validação de uma Assinatura Digital:

- Os valores no par [r,s] estão dentro do intervalo [1,n]? Se não, [REJEITA];

  Sim, 3 e 5 estão no intervalo [1,37].

- Calculamos [e] = H(m);

  Função de hash já aplicada. [e] = 17;

- Calculamos [w] = [s]-1 mod [n];

  Calculamos [w] = (1/5) mod 37 = 15;

- Calculamos $[u_1]$ = [e][w] mod [n];

  *Calculamos $[u_1]$ = 17*15 mod 37 = 33;*

- Calculamos [u_2] = [r][w] mod [n];

  *Calculamos [u_2] = 3*15 mod 37 = 8;*

- Calculamos [Z] = [$u_1 * P$] + [$u_2 * Q$];

  Calculamos [Z] = 33(1,5) + 8*(24,22) = (15,2) + (2,6) = (3,1)*

- O ponto [Z] é o Ponto no Infinito [O]? Se sim, [REJEITA];

  [Z] não é o Ponto no Infinito [O];

- Convertemos a coordenada [x_1] de Z(x_1, y_1) para um inteiro [X_1];

  Convertemos para X=3;

- Calculamos [v] = [X_1] mod [n];

  Calculamos [v] = 3 mod 37 = 3;

v =[r]? Caso positivo, [ACEITA]; Caso negativo, [REJEITA];

Temos que [v]=3 e [r]=3. Como [v]=[r], aceitamos a Assinatura Digital.

3 CENÁRIO E PARÂMETROS

3.1 Caso de Uso

De modo a melhor definir as especificações do produto, vamos gerar um cenário para um caso de uso no qual este coprocessador deve ser utilizado. Estabelecemos na introdução deste relatório que gostaríamos de uma solução de segurança para compras online através de um smartphone, então partimos disso para imaginar um sistema de pagamento que seja adequado ao nosso caso de uso:

Contexto

- Imaginamos um novo sistema de pagamentos eletrônicos. Este sistema contém uma base de dados com possíveis variações de um domínio de curvas [D];
- A cada pessoa física ou jurídica cadastrada no sistema é atribuído um código referente a um dos domínios [D] utilizados pelo sistema;
- A cada período de tempo pré-determinado, a base de dados de [D] é atualizada, e cada cliente recebe um novo código referente a seu novo domínio atribuído.
- O sistema de pagamentos apresenta uma função que gera um número inteiro, e chave privada, [d] (nunca gravado no sistema) a partir de uma senha de 4 dígitos e dados biométricos do usuário.
- Ao se cadastrar no sistema com a senha e os dados biométricos, uma chave pública [Q] será gerada a partir de [dP]. Esta chave pública passa a ser o principal

método de identificação do usuário no sistema.

Cenário

- Usuário faz uma compra em uma loja virtual através de seu smartphone, gerando, através do site, um comprovante com detalhes da compra. O comprovante passa então por uma função de hash, resultando no código [h].
- Usuário utiliza a senha e biometria para calcular [d] na hora. Smartphone repassa ao coprocessador criptográfico seu domínio [Dx], o código [h], e o inteiro [d]. Com esses dados, o coprocessador gera uma assinatura digital [A] para o comprovante e repassa ao smartphone.
- Durante a utilização pelo usuário, o coprocessador deve ter foco no tempo de resposta, já que o usuário espera terminar de maneira relativamente rápida o seu pedido.
- Loja recebe e confirma o pedido de compra ao usuário. Assinatura digital é então colocada na fila com um lote de outras assinaturas digitais, cada uma associada ao código de uma compra na loja virtual.
- O coprocessador recebe essas assinaturas em lotes, junto com as chaves públicas e domínio, e repassa ao servidor os códigos [h] decifrados de cada compra. O servidor então compara cada um dos códigos [h] com aqueles gerados por seus próprios comprovantes.
- Caso os códigos de uma compra sejam iguais, o pagamento é autorizado pelo sistema, e a compra pela loja virtual pode ser consumada.
- Durante a utilização pela loja, o coprocessador deve ter foco na vazão, já que há um número elevado de pedidos esperando autorização, e estas não precisam ocorrer imediatamente após o pedido ser efetuado.

3.2 Parâmetros de Operação

Apesar de termos estabelecido um caso de uso completo para o sistema na forma de um sistema de pagamentos, o foco de nosso projeto é o coprocessador, e não o sistema completo. Dessa forma, é necessário lembrar que não estaremos trabalhando com cálculos de biometria ou métodos de segurança para o software. O software companheiro a ser desenvolvido tem como objetivo apenas alimentar o coprocessador com dados válidos. Da mesma forma, respeitando o escopo deste projeto, vamos trabalhar com parâmetros de uso pré-estabelecidos. Isso é importante neste projeto por 2 razões:

- Podemos pular algumas etapas de cunho teórico que não são parte do escopo do projeto, mas que tomariam um tempo considerável, como geração e validação de curvas de coeficientes aleatórios através de uma semente [S];
- Podemos mais facilmente estabelecer benchmarks com as curvas mais utilizadas, bem como mais facilmente transitar entre aplicações de hardware e aplicações de software para o cálculo de curvas elípticas.

Estabelecidas estas condições, podemos definir os parâmetros que vamos utilizar neste projeto. Estamos trabalhando com campos de corpos finitos do tipo [FP], e nesse contexto, escolhemos uma curva recomendada pelo padrão FIPS 186-2 do tipo P-256 ([$F2^233$]), com os seguintes parâmetros (HEKERSON; MENEZES; VANSTONE, 2004) :

Curva P-256

- Ordem <q> = 2256 - 2224 + 2192 + 296 - 1;
- Coeficientes
  * <a> = -3;
  * <b> = 0x 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B

- Ponto P(x,y) com coordenadas:
  - <x> = 0x 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB3AA0 F4A13945 D898C296
  - <y> = 0x 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5
- Ordem do ponto P(x,y):
  - <n> = 0x FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
- Cofator <h> = 1;

Além do domínio, há outros 3 tipos de parâmetros que precisamos definir: O formato da mensagem [m] que será assinada, a função de hash [H] a qual a mensagem será submetida, e os pares de chaves pública e privada a serem distribuídos.

Formato da Mensagem

O formato de mensagem padronizado que utilizaremos para este projeto é um comprovante simples com alguns dados básicos da compra simulada em questão. Este comprovante será gerado pelo próprio software companheiro, e é da forma:

```
########################################
PCS2501 - TRABALHO DE CONCLUSÃO DE CURSO
COMPROVANTE DE COMPRA
** CÓDIGO DE COMPRA: 72094157209415
** COMPRADOR: DENNIS T S CUFLAT
** LOJA: ESCOLA POLITÉCNICA USP
** DATA DA COMPRA: 22/05/2015
** VALOR DA COMPRA: R\$100,00
########################################
```

Função de Hash

A escolha de uma função de hash [H] deve ser feita com cuidado. Ao mesmo tempo em que sua aplicação não é o papel principal do nosso dispositivo de hardware, e, portanto, poderíamos fazer uso de uma operação menos complexa por não julgar que esta faz parte do escopo do projeto, é necessário lembrar que, como estamos lindando com um cenário de segurança, não podemos nos colocar em uma situação em que tomamos conclusões erradas a respeito da validade de uma assinatura digital por conta de uma colisão de hash, o que prejudicaria todo o projeto e documentação. É importante, por isso, que a função de Hash cumpra com seus requisitos principais de criptografia para assegurar bons resultados de testes com o coprocessador:

- Resistência à Inversão: Se temos uma função de hash H, tal que geramos um código [x] através de [x] = H(m), a função de hash deve garantir que não seja possível encontrar o valor da mensagem [m] que gerou o código [x]

- Resistência à Colisão: Outra característica importante deve ser a minimização de colisões, ou seja, de mensagens [m] que resultem em um mesmo código [x]. O requisito é que, para um [m] qualquer, deve ser computacionalmente inviável calcular um outro [m] que gere o mesmo código [x]. Como estamos lidando com um modelo de comprovante pré-definido, também reduzimos as chances de encontrar um outro comprovante válido que leve a uma colisão.

Uma possível função de Hash ideal a nossos propósitos é a Keccak, vencedora de uma competição da NIST com o propósito de definir o padrão SHA-3 (Secure Hash Algorithm). Escolhemos utilizar essa função por ser conhecida e possuir alto desempenho em hardware. Além disso, fazendo uso de bibliotecas pré-estabelecidas e testadas, diminuimos o tempo de projeto.

Pares de Chaves

Podemos utilizar ferramentas de software para gerar pares de chaves a serem usados para testes com o nosso coprocessador. Utilizando a biblioteca Cryptopp, geramos as seguintes chaves de exemplo:

Chave Privada	Chave Pública (x)	Chave Pública (y)
b84dc782607e923d 2f223251c1862f0c 4f07e134c6c11707 79a8c97a290bd03e	c91b8a29d28a4042 1ee980bd581b8a9b 160d2c2846943e29 c42dd2211640d37d	fb2d7fee9fe298ca 34f98fe183161d05 b827abaeb1c9b7f2 f88d2134203068bd
a4fad484dce574ac fe635db3b7e2c791 0a0c8ba21170c424 63288c00f77fbb0a	e70328cb732b6687 8b8284c1bdbc8621 6425a28ab43e004b 3bae489c94f53665	7fc79aa222ae5144 8c235f5abceec436 ebed98926dfead6f 1fbfe5f31959fde8
2f86b8ec79c5bd0e 3b67cfacd925c2f5 4b037ec3e8360141 970b770a84c472d5	adb848cc89423648 b114f3857c9c8486 f67097b1a739b52d 4c0d10b189d52fb6	cbb666c79b38b906 570e29f2d70173c1 c4263f22c56647ba 664f1ed1860bc4e6

Figura 8: Pares de chaves de teste

4 RECURSOS

Vamos agora nos focar nos recursos que temos em mãos para realizar a síntese, simulação e testes dos módulos do nosso projeto. Iniciaremos com as ferramentas que usaremos para desenvolver o software companheiro, e depois detalharemos aquelas que, de fato, vamos utilizar para projetar nosso dispositivo de hardware.

4.1 Visual Studio 2015

Nosso software companheiro será desenvolvido com o paradigma de Orientação a Objetos. Será um software simples, que basicamente simula o sistema de pagamentos e lida com entradas e saídas para o coprocessador.

Figura 9: Visual Studio 2015

Inicialmente, o projeto especificava a linguagem Java como linguagem de programação de escolha para o software, por ser uma das linguagens mais utilizadas pra orientação a objetos, e, por isso, com amplo material online e diversas ferramentas conhecidas e amplamente difundidas. No entanto, posteriormente optamos por fazer uso de C++, por conta dos seguintes motivos:

- C++ permite que a nossa simulação das operações criptográficas em software (para benchmarks) sejam feitas nas mesmas ferramentas em que estamos desenvolvendo o software companheiro. Não poderíamos tomar esta abordagem em Java pois, como esta roda sobre o JVM, o desempenho do software como parâmetro de comparação seria prejudicado.

- C++ contém uma excelente biblioteca de criptografia, a biblioteca Crypto++, da qual podemos fazer uso nas simulações de operação do coprocessador, e também em alguns cálculos específicos em que a implementação no coprocessador não é tão vantajosa, como a operação de mapeamento.

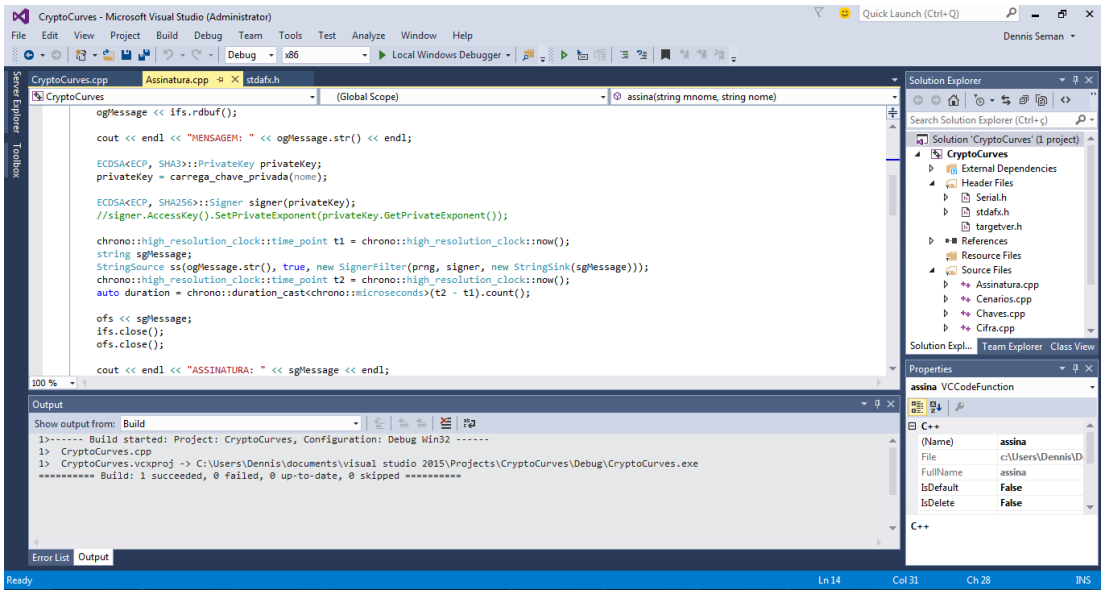


Figura 10: Interface do Visual Studio

4.2 Xilinx ISE

Durante a maior parte do desenvolvimento deste projeto, estaremos escrevendo em linguagem de descrição de hardware VHDL. Há diversas ferramentas robustas que permitem sintetizar e simular um projeto deste tipo: Durante o curso, por exemplo, já fizemos uso do Active-HDL para a síntese de um processador com pipeline.

Figura 11: Xilinx ISE Design Suite

Escolhemos a ferramenta ISE Design Suite, da Xilinx, para este projeto, pois a sua implementação prevista será com FPGAs, e o ISE oferece um ambiente completo para lidar com este tipo de implementação nos mais diversos parâmetros de dispositivos (XILINX, 2009).

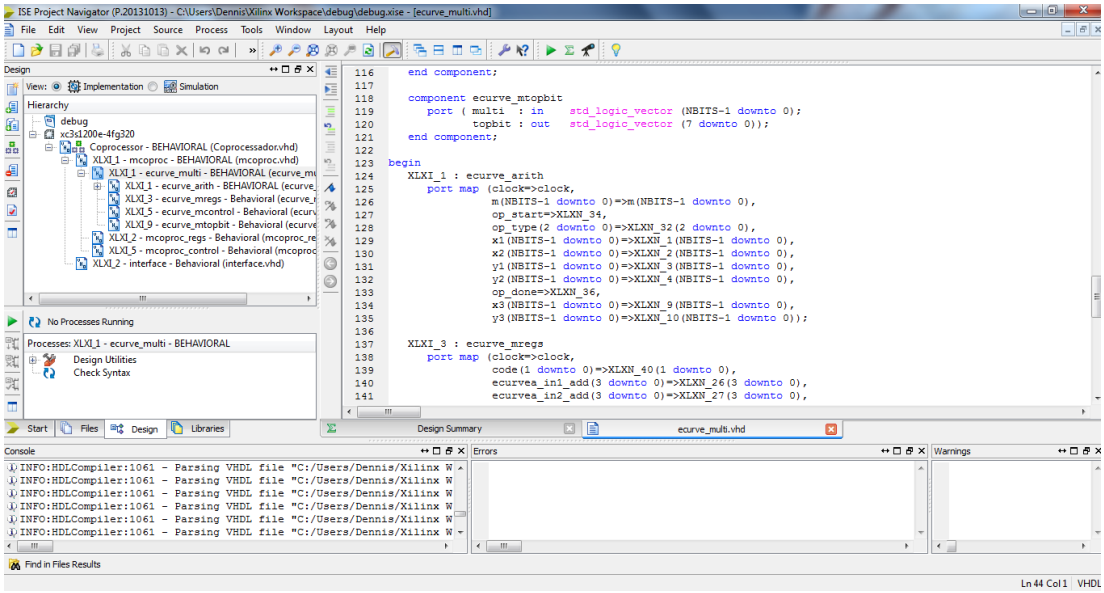


Figura 12: Interface do Xilinx ISE

5 ESPECIFICAÇÃO DE SOFTWARE

5.1 Visão Geral

Faremos agora uma especificação mais detalhada do software companheiro que será desenvolvido junto ao coprocessador. Sua função, como explicado anteriormente, será servir como uma interface entre usuário e hardware, alimentando o último com dados já verificados e no formato adequado para serem manipulados; também receberá os dados resultantes do coprocessador e os formatará de forma a serem apresentados ao usuário. Especificamos dois casos de uso que estaremos trabalhando: Geração e validação de assinaturas digitais. Para isso, é importante que o software possa servir como interface aos dois módulos. Sendo este um software desenvolvido utilizando orientação a objetos, começamos analisando os elementos do projeto de modo a identificar as classes relevantes.

Classe: Usuário

Função: Atua no papel de comprador de um item qualquer em uma loja virtual. É identificado pelo nome, sua chave pública, e é o responsável por gerar a assinatura digital.

Classe: Empresa

Função: Atua no papel de responsável pela loja virtual. É identificada pelo nome, sua chave pública, e é responsável pela validação em lotes de assinaturas digitais de modo a autorizar as compras.

Classe: Comprovante

Função: É o elemento resultante da compra de um item na loja virtual. É um dos principais elementos do projeto por ser o conjunto de dados a ser submetido aos processos criptográficos do coprocessador.

Classe: Lote

Função: Basicamente, um conjunto de comprovantes que serão encaminhados juntos para validação por parte da empresa. Tem o diferencial de ser alimentado ao software por meio de arquivos de texto, e não pela interface do terminal.

Classe: Domínio da Curva

Função: Atua como uma classe estática contendo todos os parâmetros pré-determinados para a curva com a qual trabalharemos e faremos as operações criptográficas. O programa constantemente carregará informações dessa classe para repassar ao coprocessador.

Além das classes retiradas diretamente dos elementos teóricos do projeto, precisamos considerar dois outros tipos de classe: Os sistemas e as interfaces com bibliotecas. Os sistemas são aqueles que atuam como os controladores do software e suas operações.

Classe: Sistema CryptoCurves

Função: É o sistema principal do projeto, ou classe <main>, que tem uma interface diretamente com o usuário através do terminal, e pode fazer a chamada de uma operação de geração ou validação de assinatura digital.

Classe: Operação de Geração

Função: É o controlador do caso de uso em que um usuário compra um item em uma loja, gera um comprovante e sua respectiva assinatura digital.

Classe: Operação de Validação

Função: É o controlador do caso de uso em que uma loja recebe um lote de compro-

Tabela 1: Entradas e Saídas do Software

	Tipo de Entrada	*Tipo de Saída*
Comunicação com Usuário	Terminal	Terminal
Dados de Compra	Terminal	Terminal
Dados de Validação	Arquivos .txt	Arquivos .txt
Comunicação com o Hardware	Interface USB	Interface USB

vantes e passa a validar, uma a uma, as assinaturas digitais para aprovar o pedido.

Todas as classes especificadas até então serão inteiramente escritas para este programa. No entanto, também vamos utilizar algumas bibliotecas prontas para realizar operações de manipulação e transporte de dados para que possamos mais facilmente adaptá-los ao formato do coprocessador.

Classe: Biblioteca Cypto++

Função: Inclui uma ampla quantidade de operações criptográficas para fins de benchmark, e operações de curvas elípticas para realizar ações específicas, como mapeamento de pontos.

Classe: Biblioteca Hash

Função: Integra ao sistemas as funções de hash as quais será submetido o comprovante, implementadas em uma biblioteca avulsa (no caso, função de hash SHA-3).

Classe: Biblioteca Interface USB

Função: Integra ao sistema as funções que permitem realizar comunicação serial entre software e coprocessador através de uma interface USB, implementadas em uma biblioteca avulsa.

É importante também definir quais são os canais de entrada e saídas de dados deste software. Para isto, basta analisar as classes e verificar que temos 4 demandas: interface para o usuário, entrada de dados para compra, entrada de dados para validação, e comunicação com o hardware. Mais especificamente detalhadas abaixo:

Com todos os dados em mãos, passamos a encontrar as relações entre cada classe e os requerimentos que cada uma faz as outras. Desenhamos então diagramas de sequência para cada um dos casos de uso trabalhados.

5.2 Diagrama de Sequência

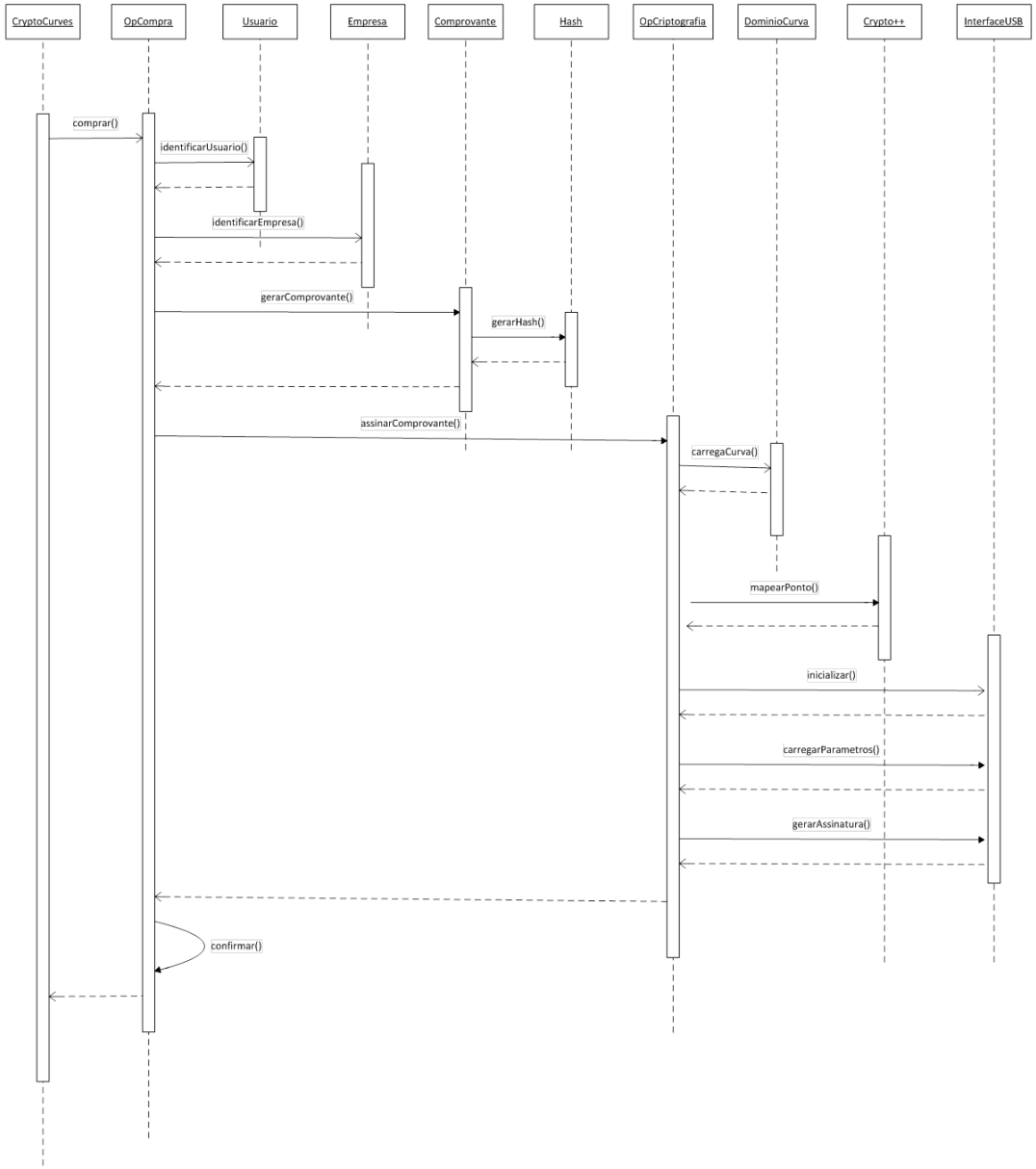


Figura 13: Diagrama do caso de uso <Gerar Assinatura Digital>

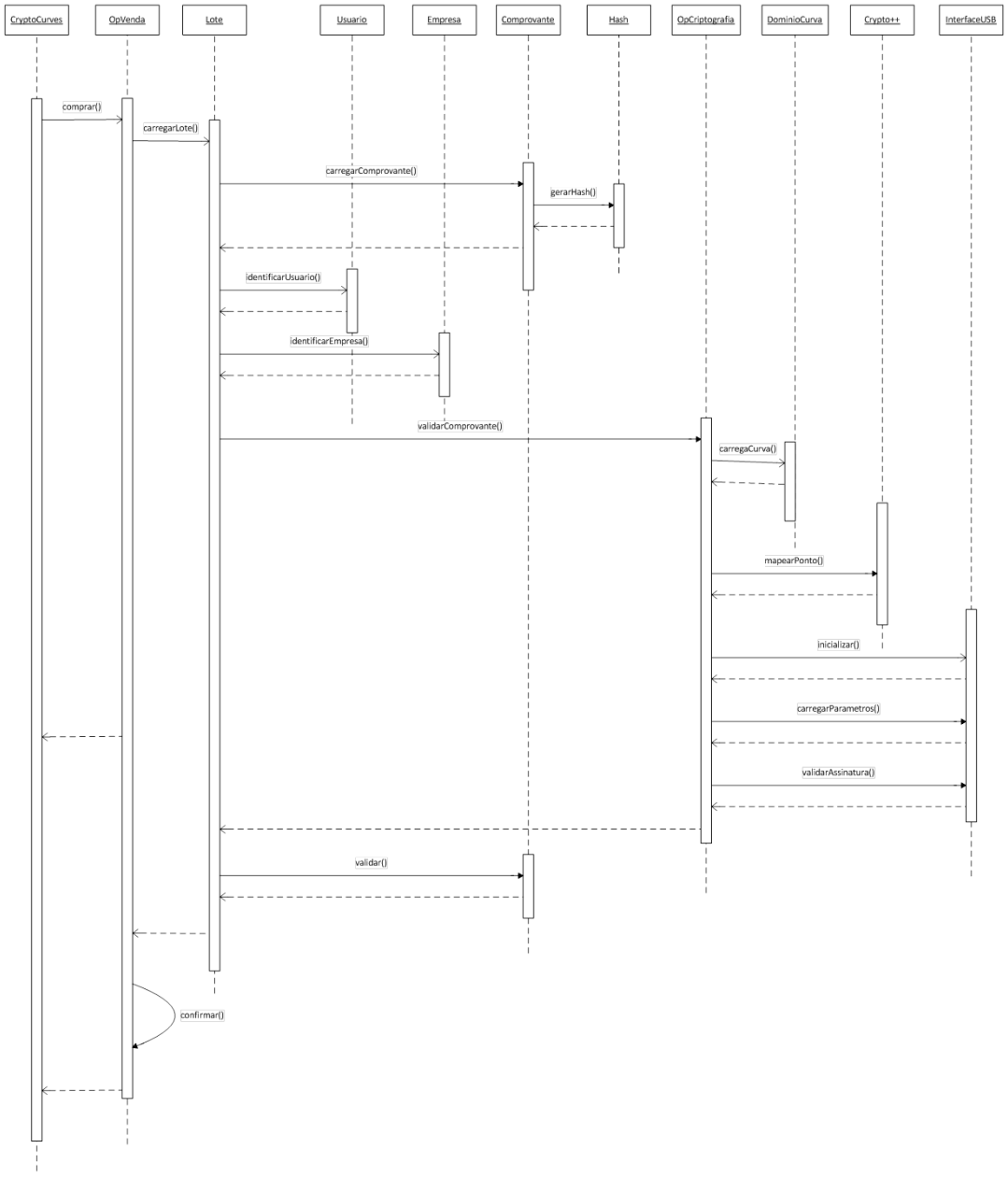


Figura 14: Diagrama do caso de uso <Validar Assinatura Digital>

6 ESPECIFICAÇÃO DE HARDWARE

6.1 Visão Geral

O foco principal deste projeto é o hardware do coprocessador, que deve ser capaz de realizar operações criptográficas a serem implementadas em um cenário de segurança, e facilmente escalável a um smartphone. Com base nos conceitos teóricos detalhados anteriormente, podemos listar todas as operações básicas que se espera fazer com este hardware. No escopo deste projeto, estas podem se apresentar na forma de 5 operações (instruções que serão passadas e reconhecidas).

- Configuração: O coprocessador deve reconhecer uma instrução de configuração, de modo a armazenar na memória todos os parâmetros de curvas a serem utilizados em seu funcionamento.
- Criptografia: O coprocessador deve ser capaz de receber um ponto da curva, e fazendo uso de uma chave pública, cifrar uma mensagem.
- Descriptografia: O coprocessador deve ser capaz de receber um ponto da curva, e, fazendo uso de uma chave privada, decifrar uma mensagem.
- Assinatura Digital: O coprocessador deve ser capaz de receber o código hash de um conjunto de dados, e a partir dele e uma chave privada, gerar uma assinatura digital.

- Validação de Assinatura: O coprocessador deve ser capaz de receber o código hash de um conjunto de dados, bem como uma assinatura digital e uma chave pública, e testar a validade dessa assinatura digital.

São, portanto, 5 operações que devem ser especificadas como instruções passadas ao coprocessador. Estas instruções serão passadas através de um canal serial por uma interface USB, que será detalhada adiante. Estas instruções compreendem processos que não podem ser realizados diretamente através de um pipeline simples. Instruções criptográficas demandam cálculos iterativos, controlados em diversas camadas de uma hierarquia de dados. A estrutura geral do coprocessador, que trabalha nestas camadas, pode ser apresentada da seguinte forma:

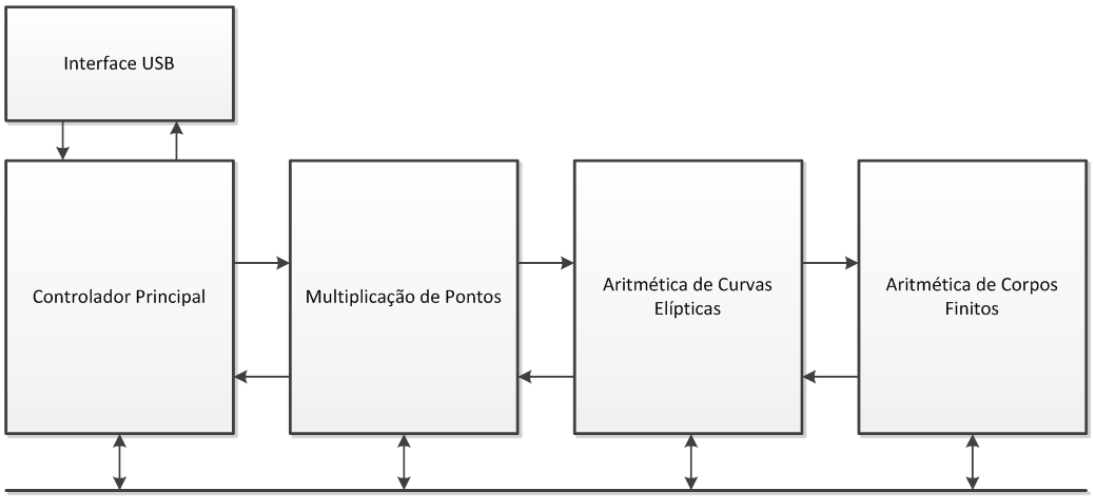


Figura 15: Estrutura do Coprocessador

6.2 Hierarquia do Fluxo de Dados

Uma vez apresentada a configuração do coprocessador, vamos detalhar um pouco a função de cada um de seus módulos. Como especificado, o hardware trabalha com uma hierarquia de dados, nos quais cada instrução demanda operações que controlam suboperações em camadas inferiores. Os dados caminham da seguinte forma:

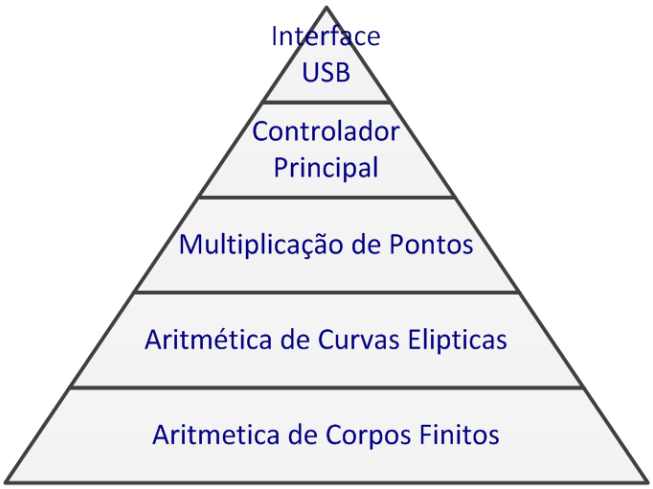


Figura 16: Hierarquia do Fluxo de Dados

Módulo Interface USB

Função: É através do módulo de interface que nosso coprocessador se comunica com o software do sistema. Este módulo tem a função de receber as instruções e seus parâmetros através de uma conexão serial, e mapeá-las em um formato que pode ser utilizado pela unidade de controle. Analogamente, tem a função de receber os outputs do coprocessador e mapeá-los de modo a permitir uma transmissão serial.

Módulo Controlador Principal

Função: Este módulo atua como a unidade de controle principal do coprocessador, recebendo as instruções da interface USB, e repassando os sinais necessários para realizá-las, incluindo subinstruções de *load* e *store* em registradores internos, gerenciamento de fila dos módulos seguintes e inserção de bolhas no pipeline.

Módulo Multiplicação de Pontos

Função: Este módulo atua na organização do processo iterativo de multiplicação escalar de pontos de curvas elípticas, que demanda operações constantes nas camadas inferiores. Esta é a principal etapa no processo de criptografia de curvas elípticas.

Operações:

$Z(x,y) = k * P(x,y)$

Módulo Aritmética de Curvas Elípticas

Função: Este pode ser considerado o módulo aritmético superior do coprocessador, e a ele compete as operações básicas de curvas elípticas, especificamente soma e reflexão de pontos, calculados através das operações geométricas demonstradas na parte teórica deste relatório. Para isso, cada operação é um processo de requisições à camada inferior de aritmética de corpos finitos.

Operações:

Z(x,y) = - P(x,y)

Z(x,y) = 2 * P(x,y)

Z(x,y) = P(x,y) + Q(x,y)

Módulo Aritmética de Corpos Finitos

Função: Esta é a camada mais inferior do sistema, responsável pelos cálculos puros de aritmética sobre os corpos finitos. Responde às camadas superiores e apresenta os processos realizados no menor número de ciclos de clock.

Operações:

Z = P mod n

Z = (P + Q) mod n

Z = (-P) mod n

Z = (P*Q) mod n

Z = (1/P) mod n

6.3 Diagrama de Instruções

Para desenvolver a unidade de controle e o fluxo de dados do coprocessador na próxima fase deste projeto, é necessário que detalhemos cada instrução, incluindo inputs, outputs, sinais e dados de memória que serão utilizados durante a sua execução. Dessa forma, desenvolvemos diagramas específicos contendo estes dados e a ordem de operação para cada instrução. De modo a facilitar o entendimento, cada etapa da ins-

trução é identificada por uma cor que marca qual camada hierárquica do coprocessador é responsável por sua execução. A tabela de cores é dada abaixo:

Figura 17: Legenda de camadas de operação

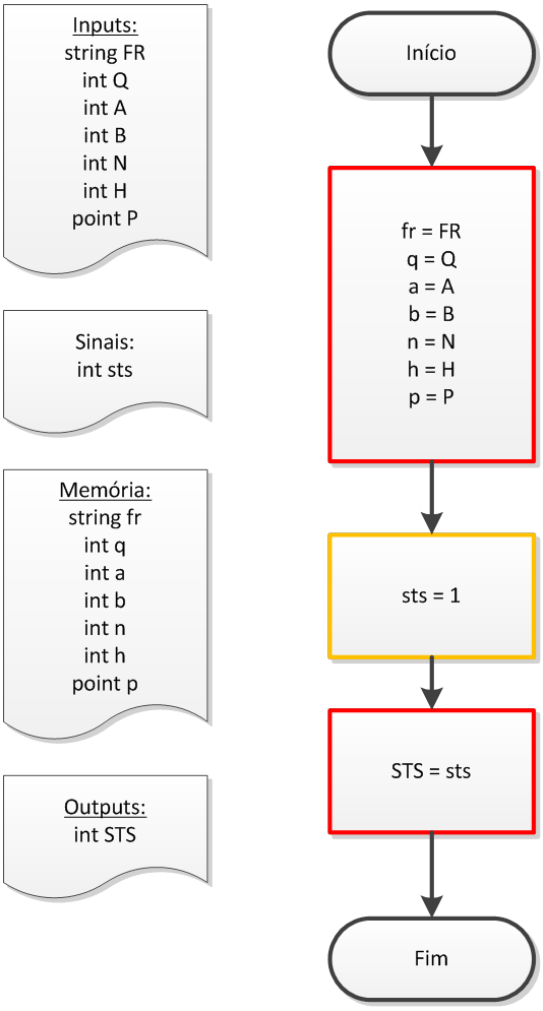


Figura 18: Instrução: configurar corocessador

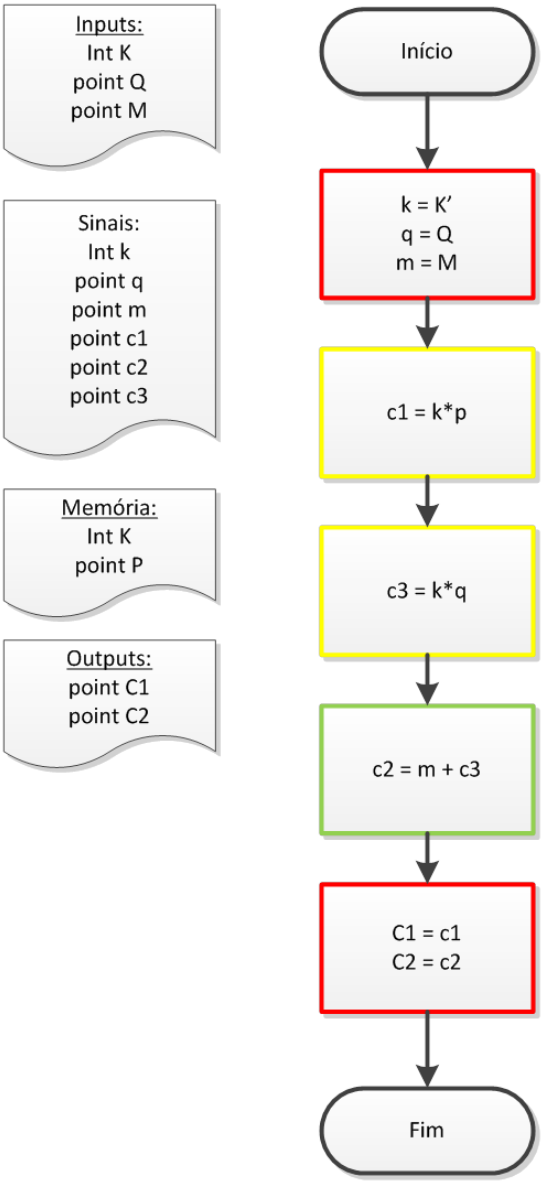


Figura 19: Instrução: cifrar uma mensagem

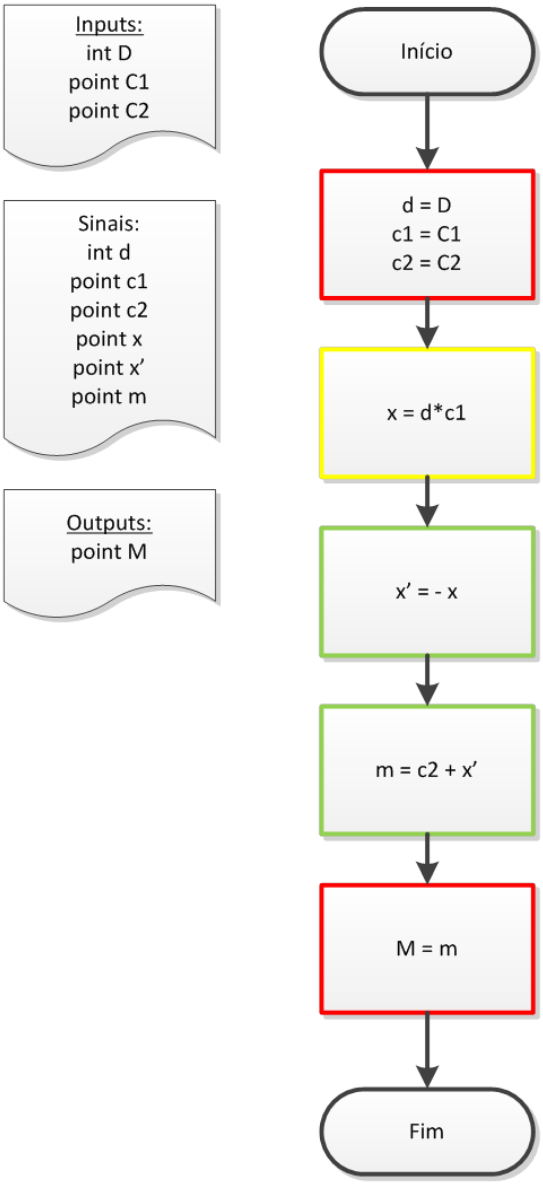


Figura 20: Instrução: decifrar uma mensagem

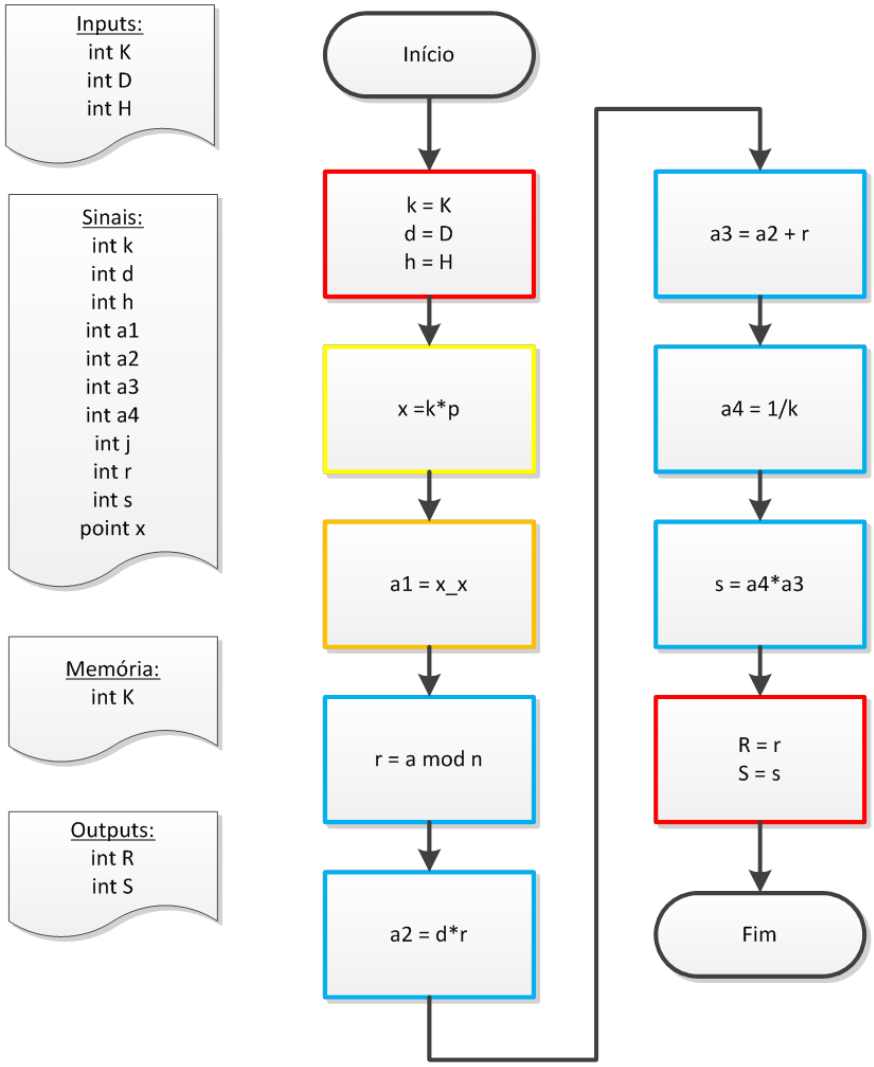


Figura 21: Instrução: gerar assinatura digital

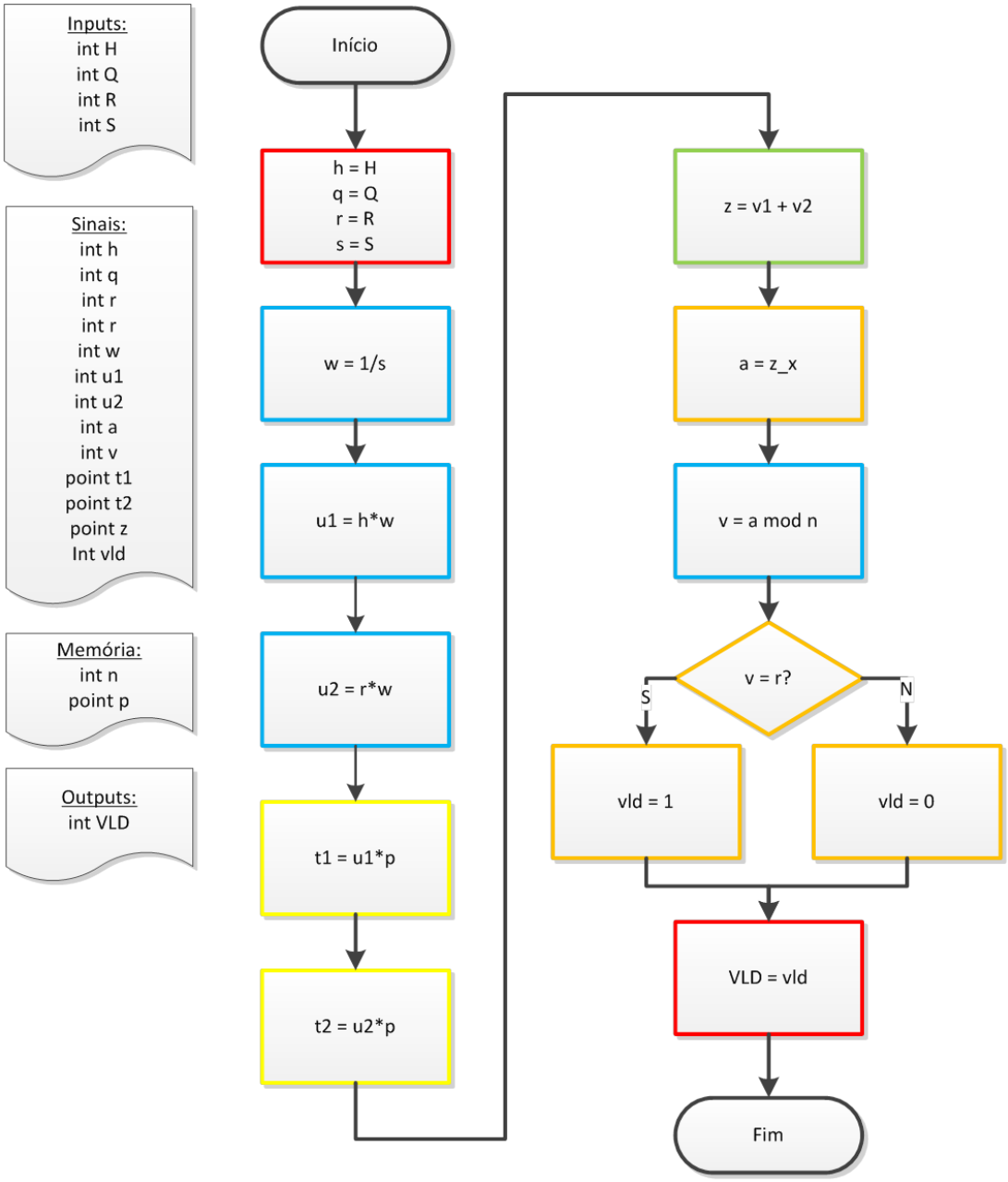


Figura 22: Instrução: validar assinatura digital

6.4 FPGA

A placa que será utilizada neste projeto é a Nexys 2, da Digilent. Ela contém a FPGA Xilinx Spartan-3E de 1200k gates (DIGILENT, 2015), ideal para este projeto. A placa também oferece diversos canais de comunicação, como USB e serial, que facilitará o desenvolvimento e depuração do coprocessador. A principal vantagem desta FPGA em nosso contexto é o fato dela ter sido desenvolvida para trabalhar com o ambiente ISE, que utilizaremos para escrever o código VHDL do coprocessador.

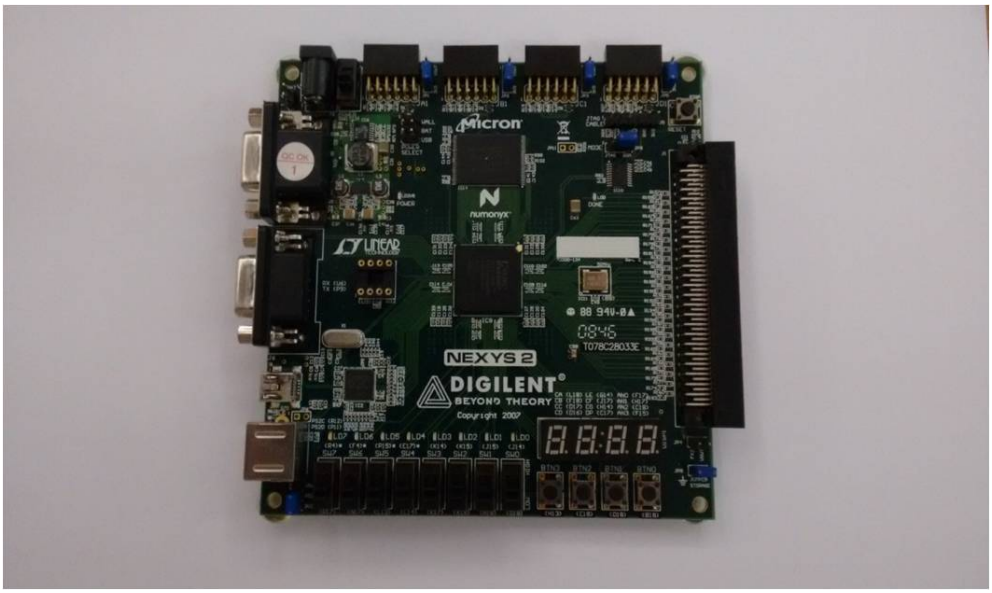


Figura 23: Placa Digilent Nexys 2

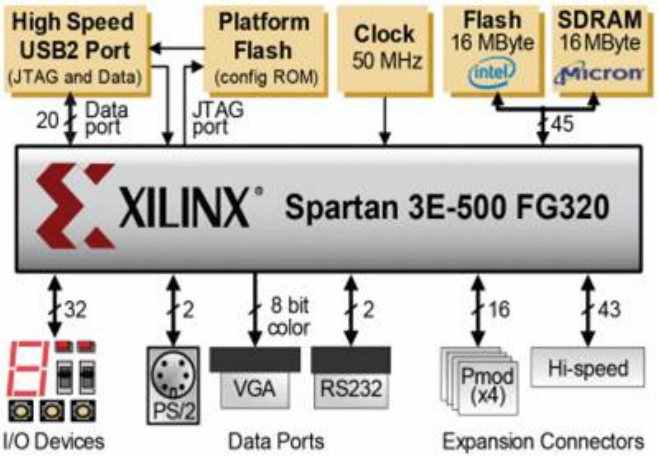


Figura 24: Especificação da Digilent Nexys 2

Para a interface entre hardware e software necessária para nosso projeto, faremos uso de um componente USB que permite comunicação serial direta, que já é suficiente para cumprir com os requisitos do projeto.

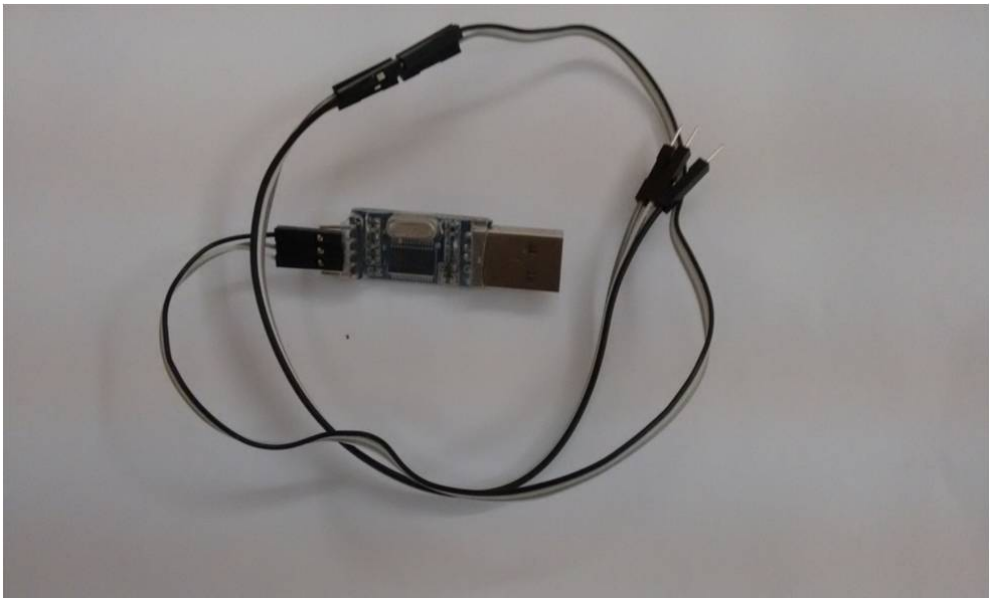


Figura 25: Interface de comunicação serial USB

7 IMPLEMENTAÇÃO DE SOFTWARE

7.1 Terminal de Comunicação

A camada de software deste projeto é composta por um aplicativo escrito em linguagem C++, capaz de se comunicar com o nosso coprocessador, e também simular suas funções a fim de gerar dados para compararmos o desempenho entre as operações criptográficas em hardware e em software. Acima de tudo, a aplicação serve como uma interface entre o usuário e nosso projeto.

Uma vez que este não era o foco deste projeto, implementamos esta camada na forma de um terminal simples, sem uma interface gráfica mais sofisticada. A estrutura interna do software sofreu algumas pequenas modificações de modo a tornar os processos mais diretos: a divisão entre classes se tornou menos dependente dos atores, e mais dependentes do fluxo do caso de uso.

Em geral, o software cumpre com todos os seus requisitos, e é capaz de realizar as seguintes funções: gerar um par de chaves, cifrar uma mensagem, decifrar uma mensagem, gerar um comprovante nos padrões especificados neste relatório, assinar um comprovante, e validar a assinatura em um comprovante. As telas abaixo acompanham os casos de uso retratados aqui.

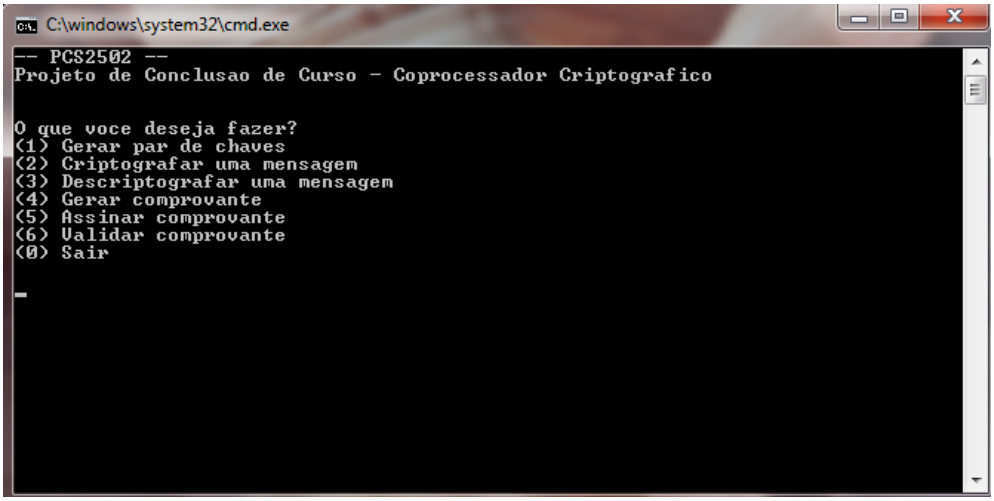


Figura 26: Interface inicial do software

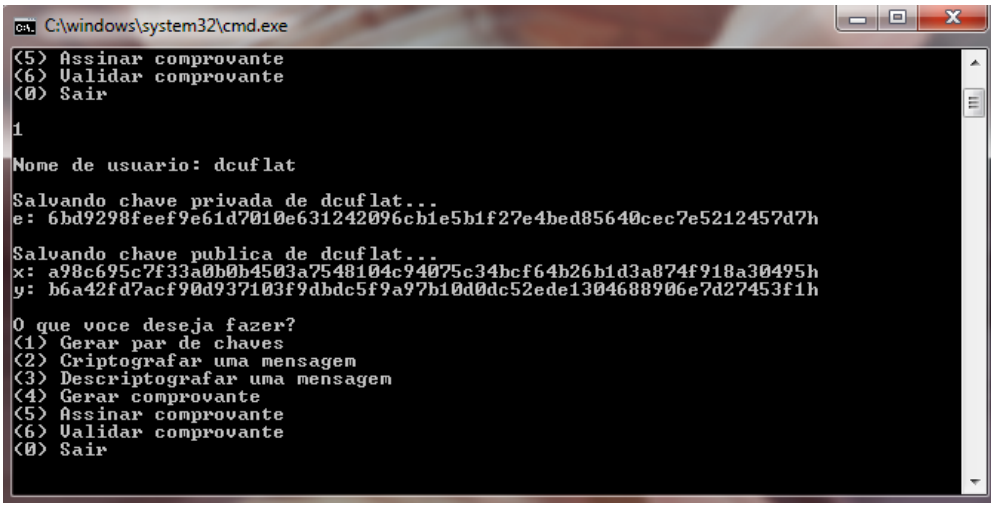


Figura 27: Caso de uso - gerando um par de chaves

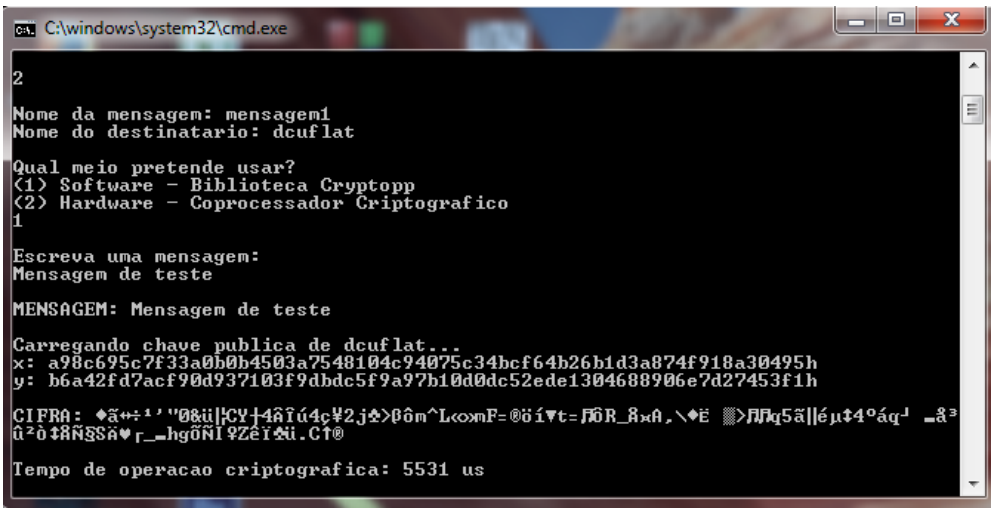


Figura 28: caso de uso - cifrando uma mensagem

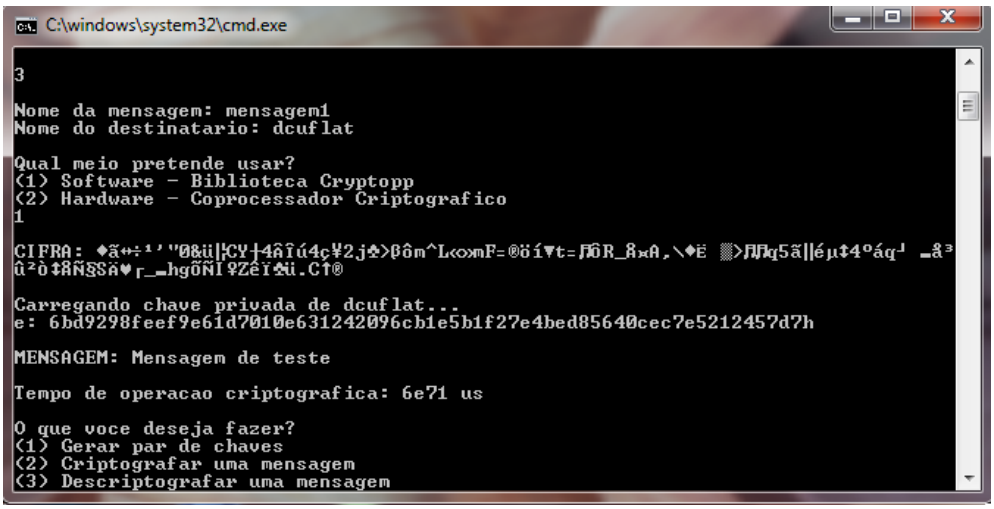


Figura 29: Caso de uso - decifrando uma mensagem

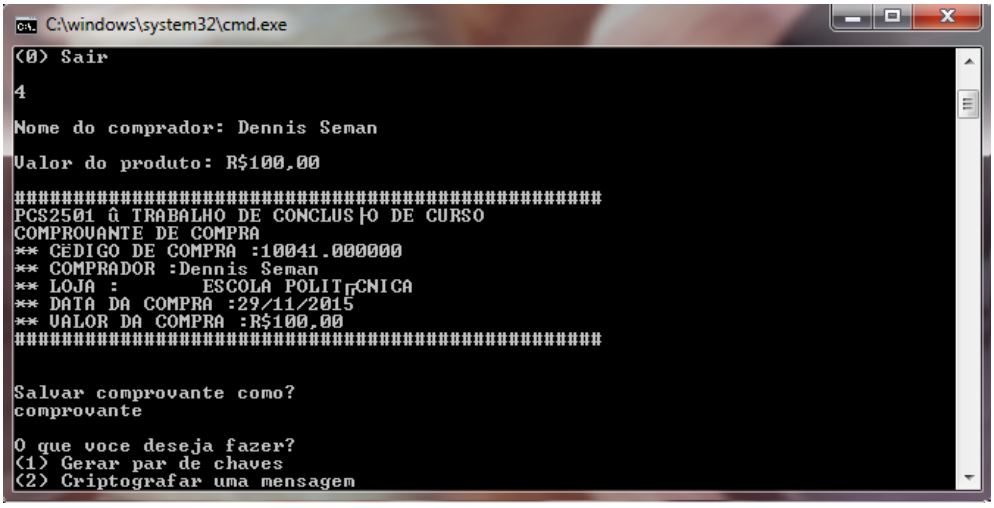


Figura 30: Caso de uso - gerando um comprovante

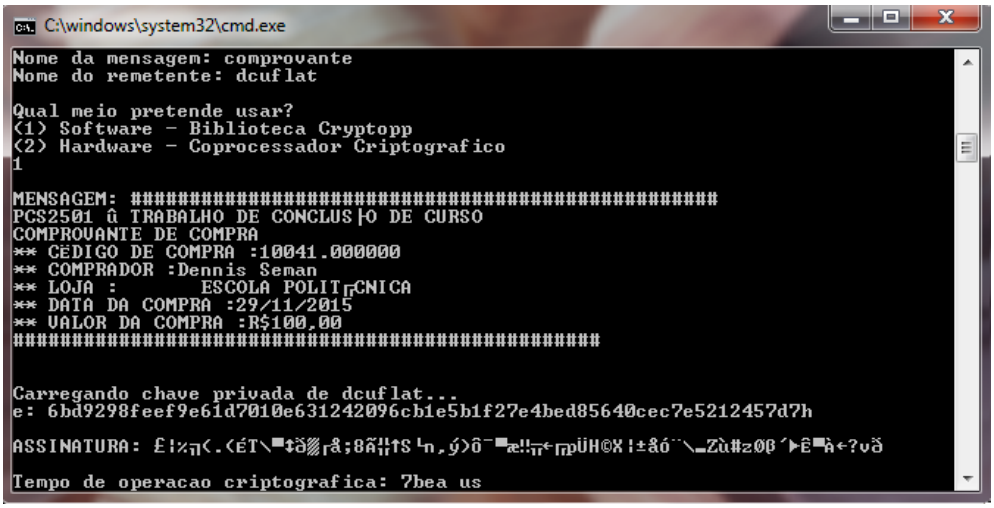


Figura 31: Caso de uso - assinando um comprovante

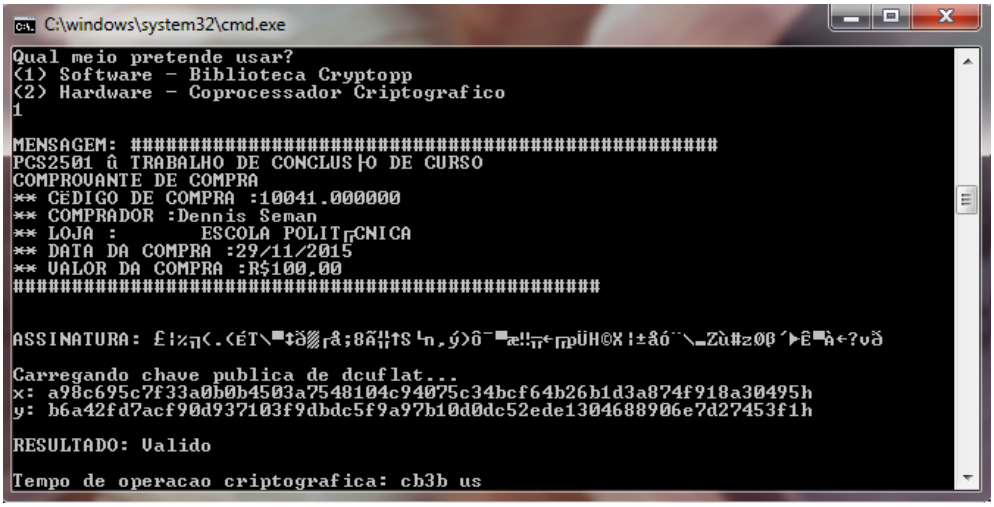


Figura 32: Caso de uso - validando a assinaturo de um comprovante

8 IMPLEMENTAÇÃO DE HARDWARE

8.1 Aritmética de Corpos Finitos

O primeiro módulo desenvolvido no projeto foi aquele que está na base da hierarquia do processador, o módulo de aritmética de corpos finitos. Basicamente todos os outros módulos implementam operações que dependem diretamente deste. A função principal do módulo é receber os parâmetros de módulo do corpo finito, tipo de operação, e dois operandos (que necessariamente devem estar dentro do intervalo definido pelo campo), e devolver o resultado da operação. Implementamos ainda uma camada inferior dentro desta, que atua como a ULA da camada para os casos em que a operação é feita em um único ciclo de clock (combinatória). São essas operações a adição, subtração, e multiplicação. Estas operações são implementadas diretamente em código VHDL, seguindo as regras e instruções definidas anteriormente neste relatório para a aritmética de corpos finitos.

Acompanham os blocos operacionais da ULA buffers de entrada e saída, que evitam constantes alterações de estado e tornam o consumo mais eficiente, e um minicontrolador responsável por estes buffers.

Com isto, montamos o nosso módulo de aritmética de corpos finitos. Já temos uma ULA para as operações combinatórias, porém ainda precisamos de um componente capaz de realizar a operação de inversão, definida pelo algoritmo euclidiano estendido para campos de Galois (GF), baseado em divisores comuns. Este componente também foi implementado diretamente em código VHDL, mas é um processo

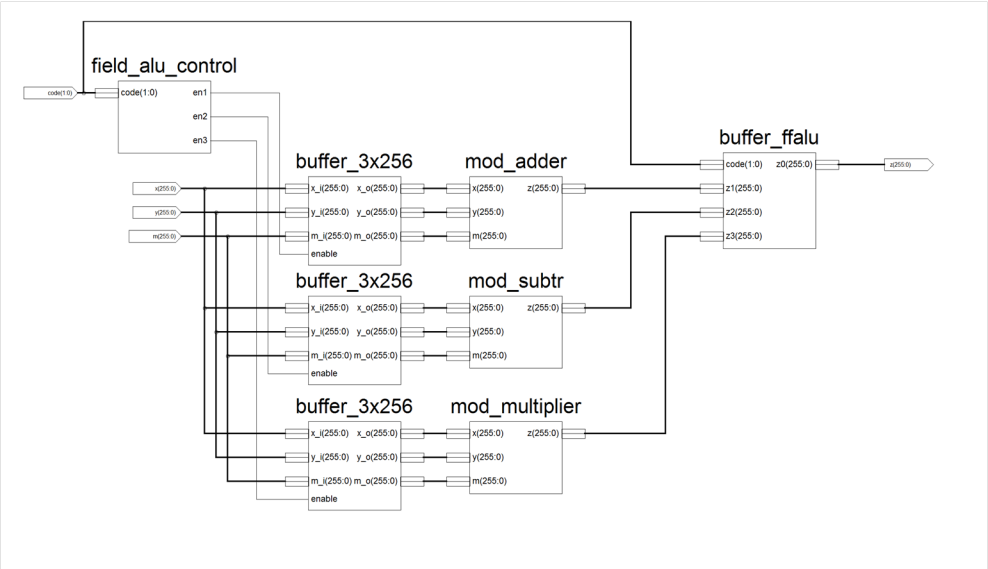


Figura 33: ULA do módulo de aritmética de corpos finitos

iterativo, que demanda um maior número de ciclos de clock, e portanto requer sinais de controle adicionais como *start* e *done*. Dessa forma, ao nosso módulo fazemos duas adições: Primeiramente, precisamos de um registrador que guarde o resultado das operações, especialmente no caso das operações combinatórias, cujos resultados permanecem corretos apenas durante um ciclo. Adicionalmente, ainda precisamos de uma unidade de controle do módulo, que implementa uma máquina de estados simples.

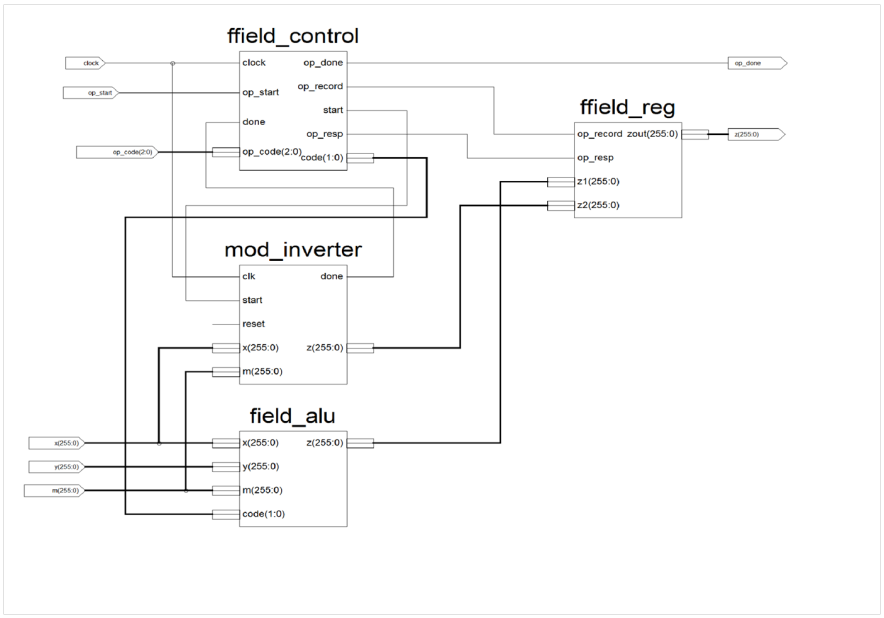


Figura 34: Módulo de aritmética de corpos finitos

A máquina de estados simplesmente contém um estado inicial de repouso, dois estados que disparam operações (combinatória ou iterativa), dois estados que aguardam resposta e comandam a gravação do resultado no registrador, e um estado que indica ao coprocessador que as operações já foram feitas e o resultado está disponível.

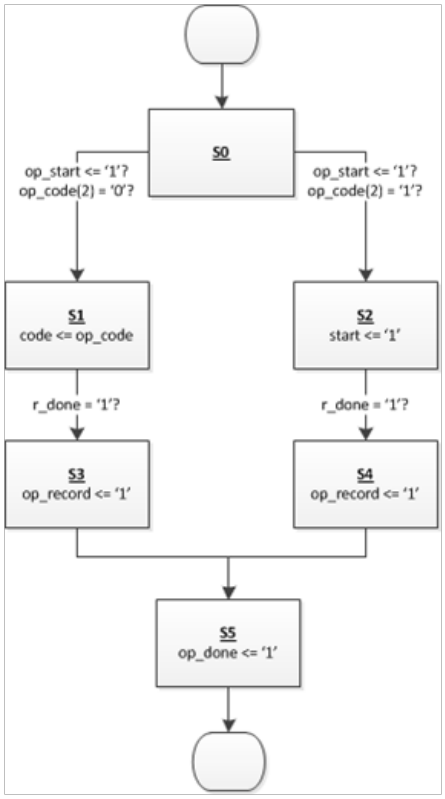


Figura 35: Máquina de estados do módulo de aritmética de corpos finitos

Depois de terminado o módulo, fazemos alguns testes para garantir que este esteja funcionando corretamente. O processador é capaz de lidar com operações com operandos de até 256 bits, porém, a fim de tornar os resultados mais compreensíveis, colocamos no relatório os testes com valores muito mais baixos. A seguir, estão as formas de ondas para os 4 tipos de operação: soma, subtração, multiplicação e inversão. Todas sobre um corpo de módulo primo 29, e apresentando resultados corretos, conforme esperamos deste módulo do projeto.

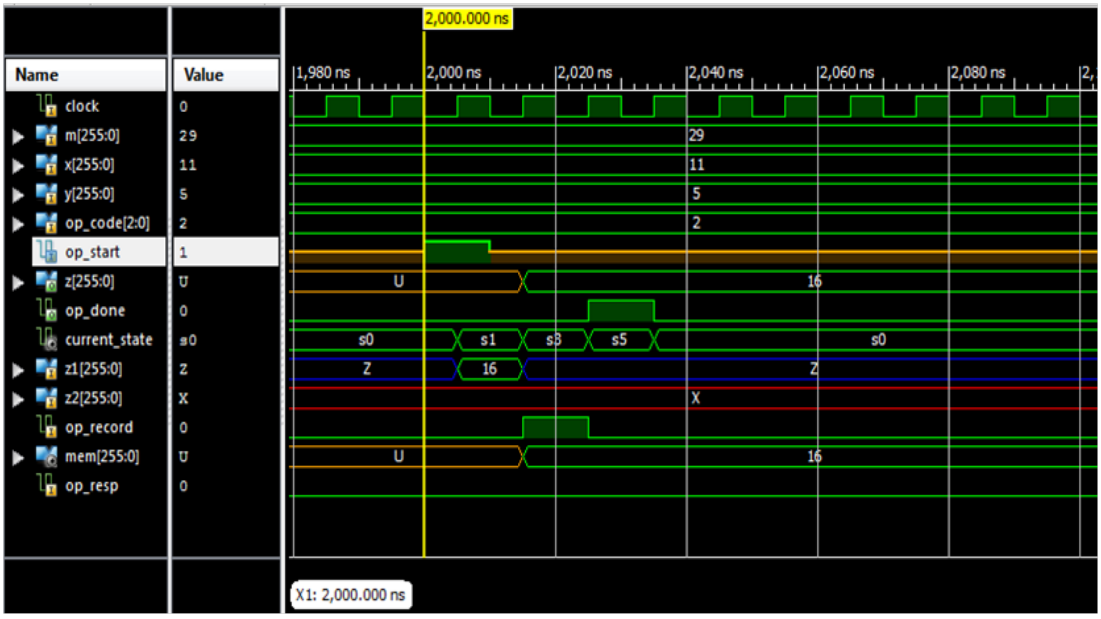


Figura 36: Simulação de uma operação de adição

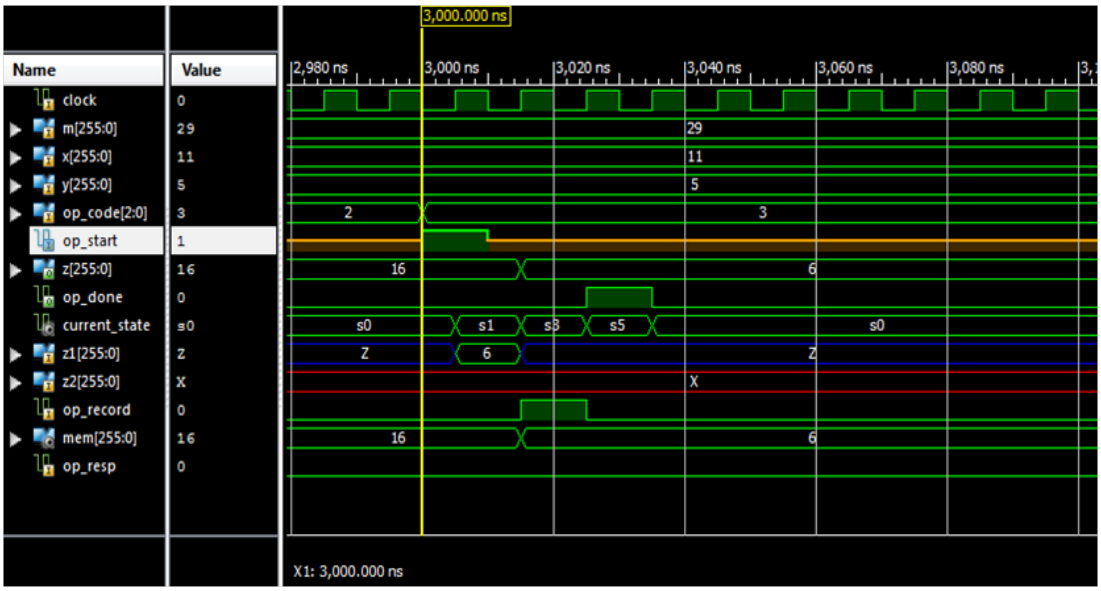


Figura 37: Simulação de uma operação de subtração

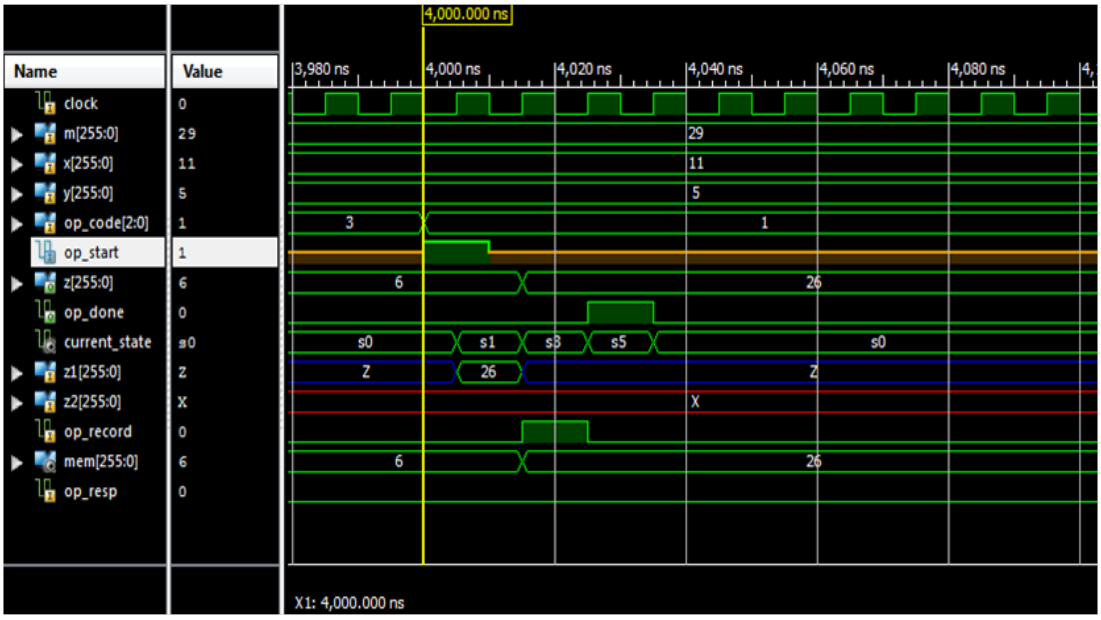


Figura 38: Simulação de uma operação de multiplicação

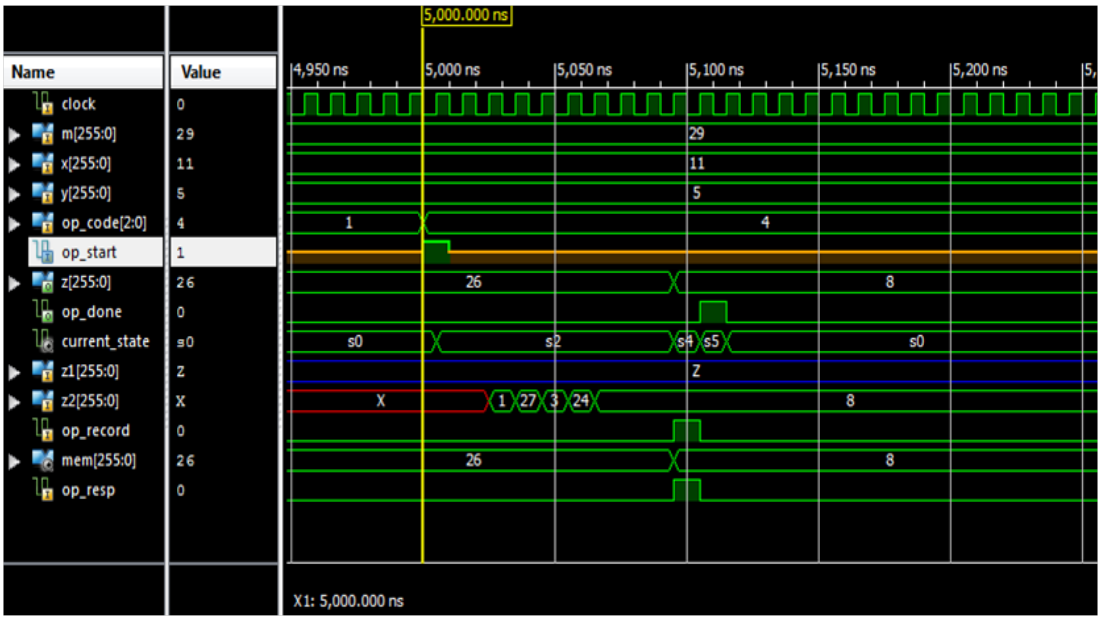


Figura 39: Simulação de uma operação de inversão

8.2 Aritmética de Curvas Elípticas

O segundo módulo desenvolvido é o módulo de aritmética de curvas elípticas. Ele implementa diretamente o módulo anterior para realizar operações sobre pontos de uma curva representada pela equação de Weierstrass, definida anteriormente neste relatório. A sua função principal é receber os parâmetros de módulo do corpo finito, tipo de operação, e dois operandos (na forma de dois pontos da curva pré-estabelecida, cada um definido por um par x,y), e devolver o resultado da operação. Vale notar que ainda é útil permitir que operações avulsas (adição, subtração, multiplicação e inversão) sejam acessadas através deste módulo, e não apenas operações de pontos. Portanto devemos levar isso em conta na hora de projetar o controlador. Temos para esta camada o módulo de aritmética de corpos finitos, o controlador, e também um banco de registradores. Como as operações sobre pontos demandam novas variáveis, precisamos de um local no módulo para armazenar os resultados de cada sub-operação. Por conta disso, além de sinais de tipo de operação, o controlador também deve manipular sinais de endereçamentos (tanto de entrada como de saída dos registradores), comandos de gravação e disparo de operações.

A nova máquina de estados deste módulo é um pouco mais robusta do que a do módulo anterior, visto que precisa implementar sub-rotinas inteiras. As duas principais operações, as realizadas sobre os pontos de curva, são a soma de dois pontos e o dobro de um ponto, que acompanham as quatro sub-rotinas que apenas chamam as operações primárias individuais. Apesar do número maior de estados, a estrutura da máquina ainda se mantém relativamente similar, com um estado inicial de repouso, os estados relativos a cada operação, e um estado final, que indica ao coprocessador o término da execução e disponibilidade dos resultados. Abaixo da máquina de estados, também inserimos duas tabelas que tratam dos sinais internos de cada passo das duas operações de pontos, incluindo os códigos de operação e endereço dos registradores utilizados em cada etapa do processo;

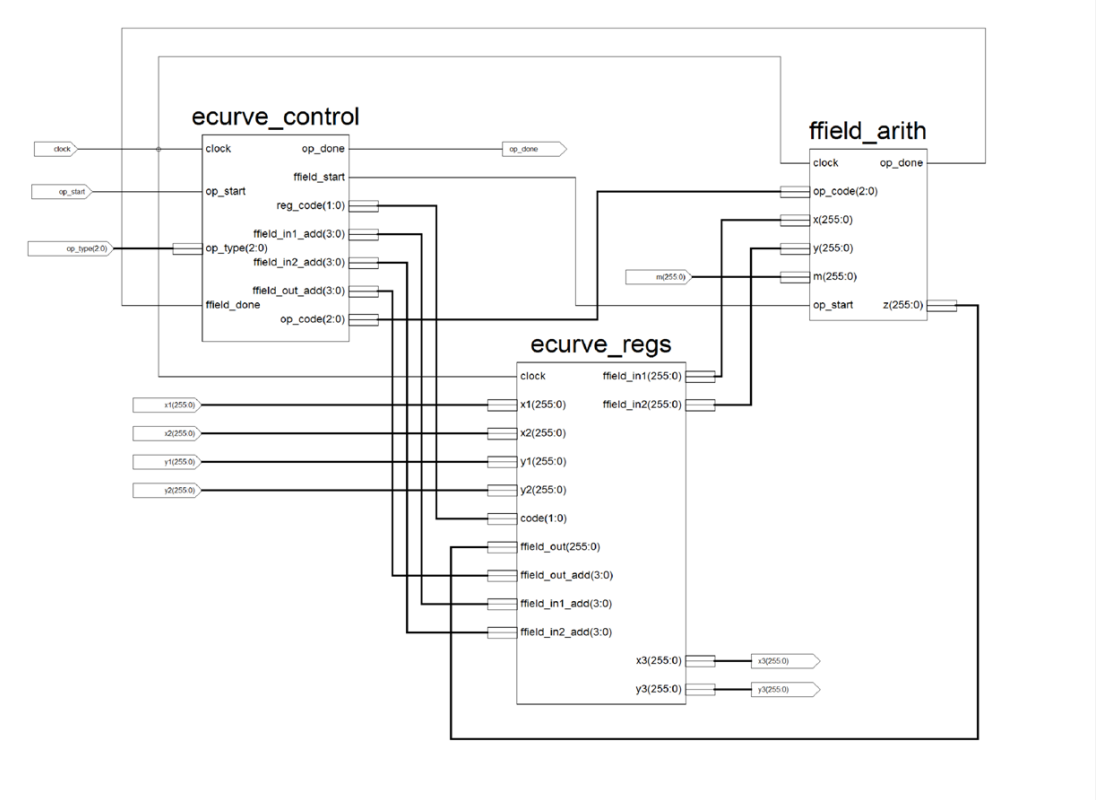


Figura 40: Módulo de aritmética de curvas elípticas

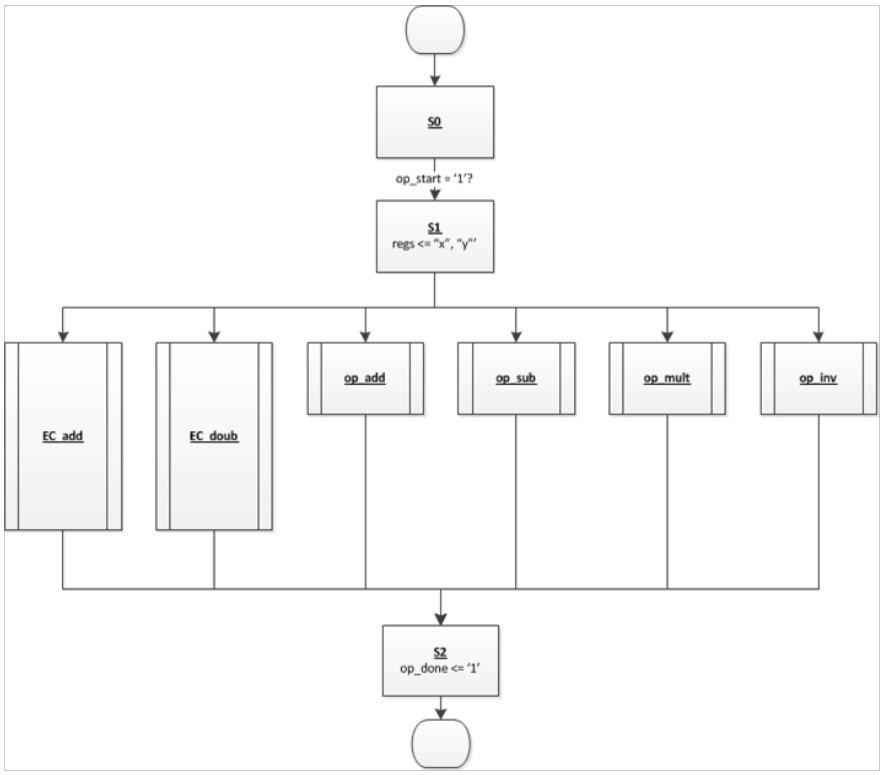


Figura 41: Máquina de estados do módulo de aritmética de curvas elipticas

CURVAS ELÍPTICAS – SOMA DE DOIS PONTOS				
$x3 = ((y2 - y1)/(x2 - x1))^2 - x1 - x2$				
$y3 = ((y2 - y1)/(x2 - x1))(x1 - x3) - y1$				
Parâmetros: X1,X2,Y1,Y2			**Reg Parâmetros:** R0,R1,R2,R3	
Cálculo	**Registradores**			**Código Operação**
	Resultado	**Operador 1**	**Operador 2**	
R1 = Y2 - Y1	4	3	2	011
R2 = X2 - X1	5	1	0	011
R2 = 1 / R2	5	5	0	100
R1 = R1 * R2	4	4	5	011
R2 = R1 * R1	5	4	4	011
R3 = X1 + X2	6	0	1	010
R3 = R2 - R3	6	5	6	011
R4 = X1 - R3	7	0	6	011
R2 = R1 * R4	5	4	7	001
R4 = R2 - Y1	7	5	2	011
Saídas: X3,Y3			**Reg Saídas:** R6,R7	

Figura 42: Estados - Soma de dois pontos

CURVAS ELÍPTICAS – DOBRO DE UM PONTO				
$x3 = ((3x1^2 + a)/(2y1))^2 - 2x1$;				
$y3 = ((3x1^2 + a)/(2y1))\ (x1 - x3) - y1$				
Parâmetros: X1,A,Y1,3			**Reg Parâmetros:** R0,R1,R2,R3	
Cálculo	**Registradores**			**Código Operação**
	Resultado	**Operador 1**	**Operador 2**	
R1 = X1 * X1	4	0	0	001
R1 = 03 * R1	4	3	4	001
R1 = R1 + 0A	4	4	1	010
R2 = Y1 + Y1	5	2	2	010
R2 = 1 / R2	5	5	0	100
R1 = R1 * R2	4	4	5	001
R2 = R1 * R1	5	4	4	001
R3 = X1 + X1	6	0	0	010
R3 = R2 - R3	6	5	6	011
R4 = X1 - R3	7	0	6	011
R2 = R1 * R4	5	4	7	001
R4 = R2 - Y1	7	5	2	011
Saídas: X3,Y3			**Reg Saídas:** R6,R7	

Figura 43: Estados - Dobro de um ponto

Tal como no módulo anterior, depois de terminada a descrição, fazemos testes para avaliar o seu correto funcionamento. Vamos demonstrar abaixo as formas de ondas para as duas operações de pontos, soma e dobro. Como referência à parte teórica deste relatório, vamos utilizar as operações que exemplificamos logo ao introduzi-las: Em um campo primo de módulo 29, com a curva da forma:

$$E : y^2 = x^3 + 4x + 20$$

Operação de Soma: (5,22) + (16,27) = (13,6);

Operação de Dobro: 2*(5,22) = (14,6);

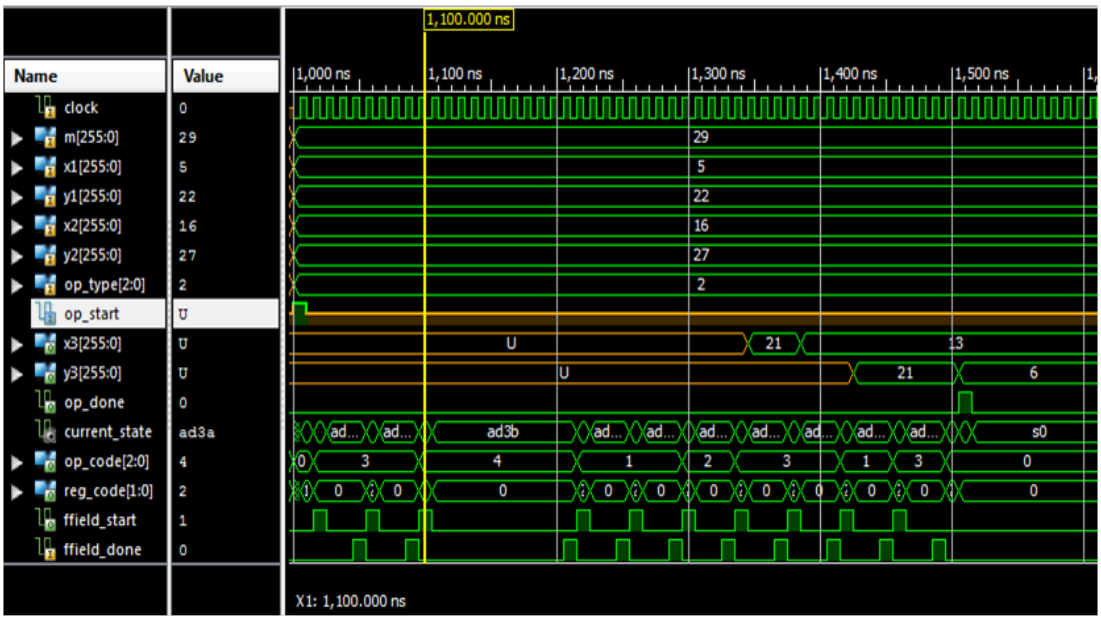


Figura 44: Simulação de uma operação de soma de pontos

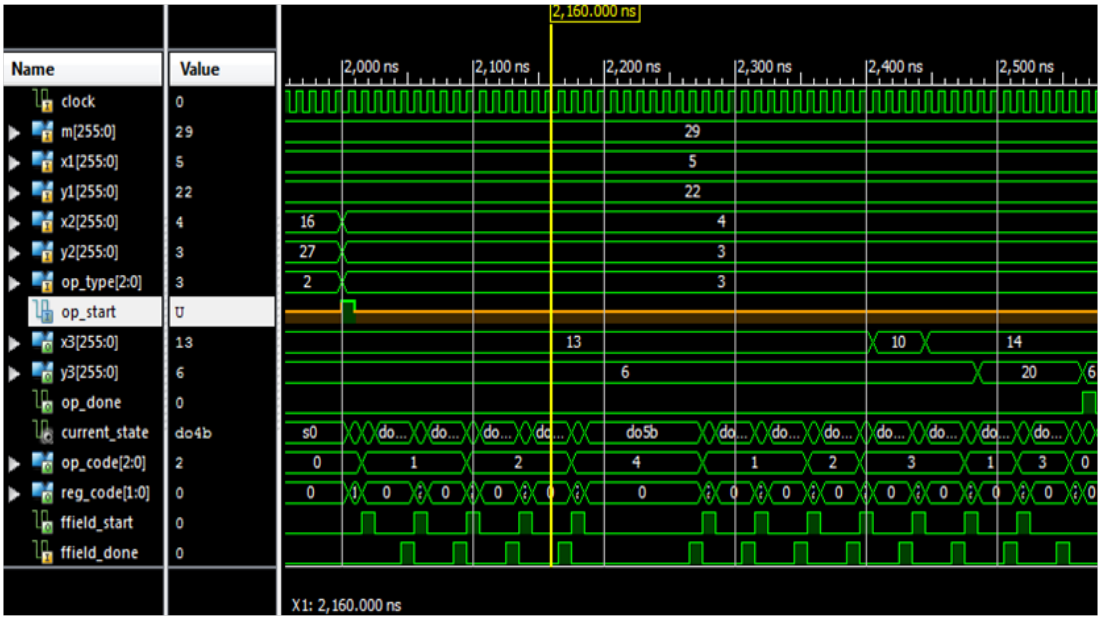


Figura 45: Simulação de uma operação de dobro de um ponto

8.3 Multiplicação Escalar de Pontos

O terceiro módulo é o módulo de multiplicação escalar de pontos para curvas elípticas. A multiplicação escalar é completamente dependente de operações sucessivas de soma de pontos, e portanto, o módulo é baseado em um loop que faz chamadas de soma conforme seus operandos. A operação é feita da seguinte forma: O ponto a ser multiplicado é dobrado e registrado em um banco interno por tantas vezes quanto os bits do multiplicador, do menos ao mais significativo, levam para atingir o 1 mais significativo (por exemplo, para uma multiplicação por 9, em binário, 1001, são 4 operações de dobro até o 1 mais significativo). Realizadas as multiplicações, são feitas as chamadas de soma de pontos para cada resultado de dobro equivalente a um bit 1 do multiplicador, até chegar ao resultado final. Resumidamente, temos uma soma parcial de pontos resultantes de operações de potência, e a estrutura do módulo repete a trinca de controlador, banco de registradores, e módulo aritmético, que verificamos nos casos anteriores.

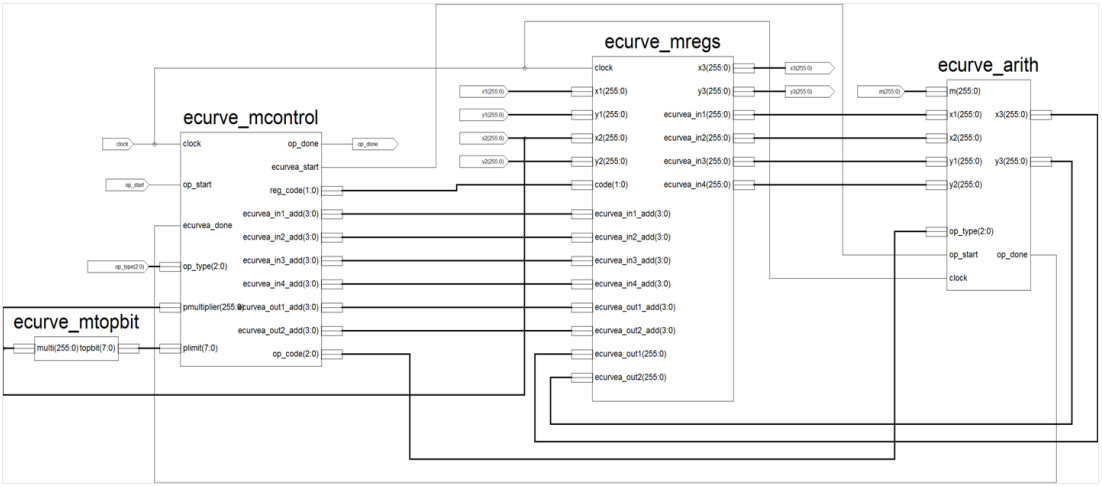


Figura 46: Módulo de multiplicação escalar de pontos

Os parâmetros da multiplicação são: ponto, multiplicador, e coeficiente <a> da curva, mas, tal como nos outros módulos, também é necessário adaptar este de modo que consiga efetuar operações avulsas de qualquer camada inferior, seja uma soma singular de pontos, ou uma operação de multiplicação sobre curvas elípticas. Sendo assim, a estrutura da máquina de estados deste módulo é praticamente a mesma daquela do módulo de aritmética de corpos finitos. Demonstramos abaixo, então, apenas a máquina de estados da sub-operação de multiplicação escalar de pontos.

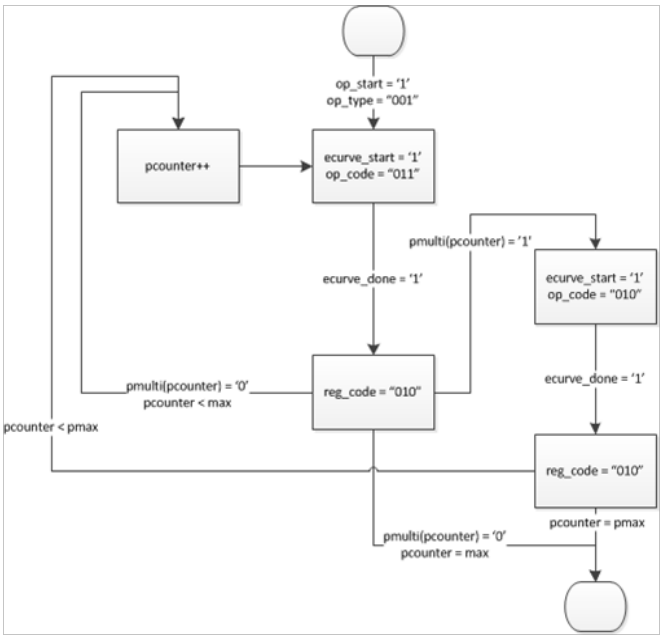


Figura 47: Máquina de estados da operação de multiplicação escalar de pontos

Tendo já este módulo pronto, demonstramos abaixo as formas de onda de uma operação de multiplicação escalar, novamente fazendo uso do ponto (5,22) da curva que estamos utilizando de exemplo, e multiplicando-o escalarmente por 9, resultando no ponto (3,1).

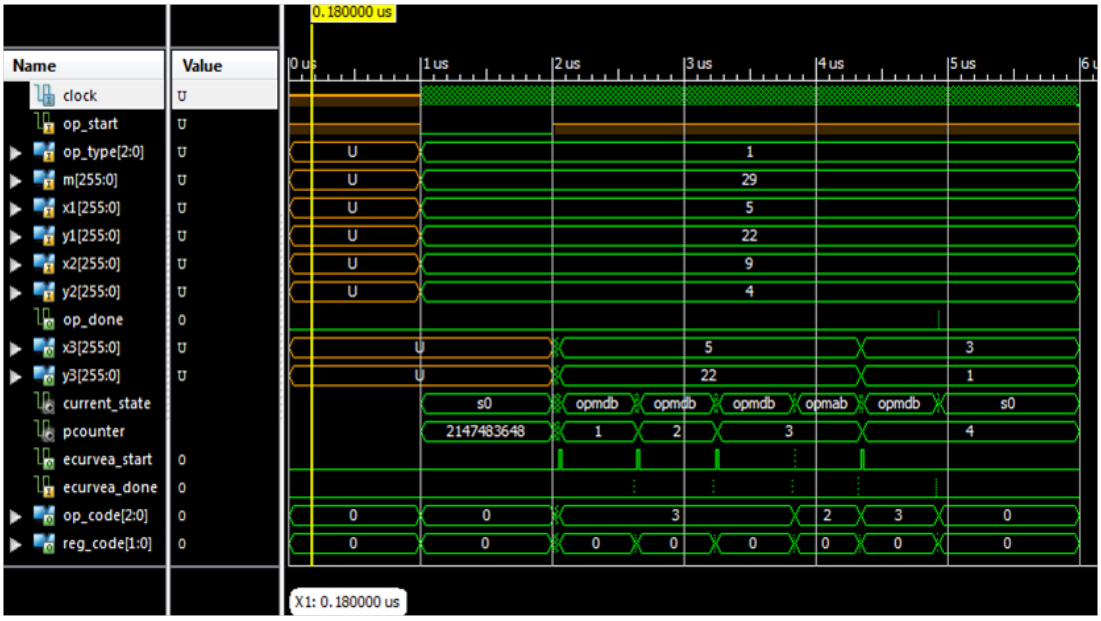


Figura 48: Simulação de uma operação de multiplicação escalar de pontos

8.4 Controlador Principal

Finalmente, temos o módulo de controlador principal, que se encontra no topo da hierarquia do coprocessador (exceto a interface de comunicação com software). Este módulo é responsável por fazer chamadas a qualquer tipo de operação implementada pelos módulos inferiores a fim de completar as 4 principais funções do coprocessador: cifrar uma mensagem, decifrar uma mensagem, gerar uma assinatura do coprocessador digital, e validar uma assinatura digital. Resumidamente, este é o módulo que controla todas as funções e implementa diretamente as máquinas de estado de nossa especificação. Ao ser iniciado, pode carregar os parâmetros encaminhados pelo software ou receber o código de operação referente a uma de suas funções e iniciar o seu respectivo processo, retornando à interface hardware-software os resultados obtidos.

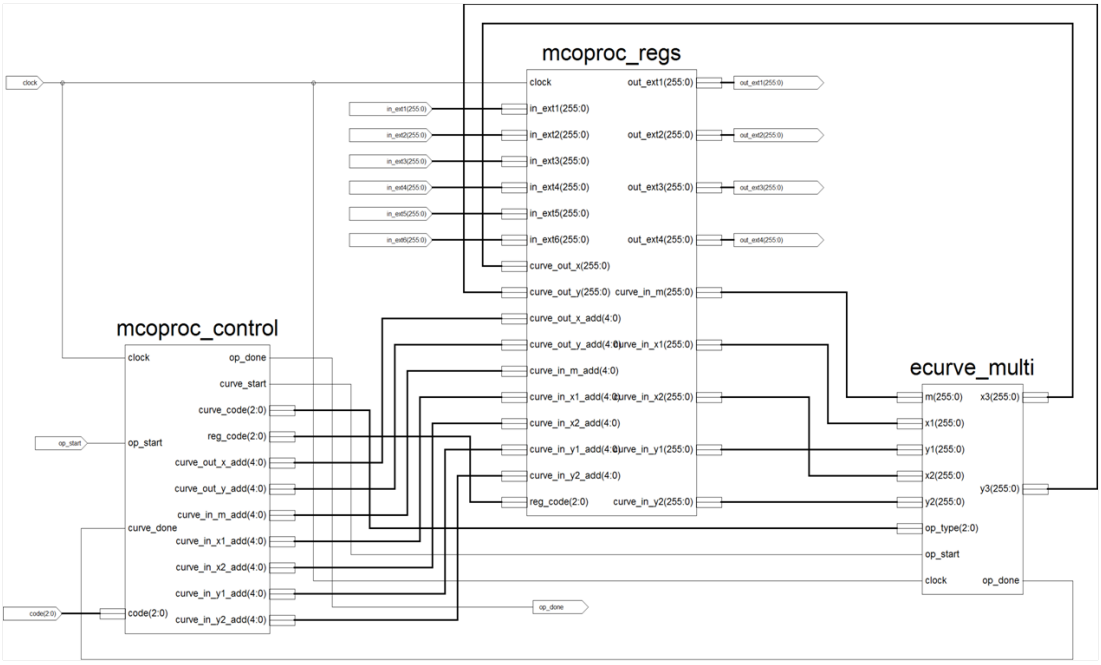


Figura 49: Módulo de multiplicação escalar de pontos

Uma vez que a máquina de estados para os processos deste módulo foram detalhadas na especificação do sistema, inserimos abaixo as tabelas que tratam de todos os passos das funções e seus devidos códigos de operação e de endereçamento no banco de registradores.

OPERAÇÃO – CRIPTOGRAFAR UMA MENSAGEM				
Entradas: Point P, Point Q, Point M, int k				
Saídas: Point C1, Point C2				
Parâmetros: Px, Py, Qx, Qy, Mx, My, k			**Reg Parâmetros:** R26, R27, R0, R1, R2, R3, R4	
Cálculo	**Registradores**			**Código Operação**
	Resultado	**Operador 1**	**Operador 2**	
(R28, R29) = R4 * (R26, R27)	28,29	26,27	4,22	001
(R0, R1) = R4 * (R0, R1)	0,1	0,1	4,22	001
(R30, R31) = (R2, R3) + (R0, R1)	30,31	2,3	0,1	010
Saídas: C1x, C1y, C2x, C2y		**Reg Saídas:** R28, R29, R30, R31		

Figura 50: Estados - operação de cifração de uma mensagem

OPERAÇÃO – DESCRIPTOGRAFAR UMA MENSAGEM				
Entradas: Point C1, Point C2, int d				
Saídas: Point M				
Parâmetros: C1x, C1y, C2x, C2y, d			**Reg Parâmetros:** R0, R1, R2, R3, R4	
Cálculo	**Registradores**			**Código Operação**
	Resultado	**Operador 1**	**Operador 2**	
(R0, R1) = R4 * (R0, R1)	0,1	0,1	4,22	001
R1 = - R1	1	20	1	101
(R28, R29) = (R0, R1) + (R2, R3)	28,29	0,1	2,3	010
Saídas: Mx, My		**Reg Saídas:** R28, R29		

Figura 51: Estados - operação de decifração de uma mensagem

OPERAÇÃO – GERAR UMA ASSINATURA DIGITAL				
Entradas: Point P, int m, int d, int k				
Saídas: int r, int s				
Parâmetros: Px, Py, Mx, My, d, k			**Reg Parâmetros:** R26, R27, R0, R1, R2, R3	
Cálculo	**Registradores**			**Código Operação**
	Resultado	**Operador 1**	**Operador 2**	
(R4, R5) = R3 * (R26, R27)	4,5	26,27	3,22	001
R28 = 1 * R4 (mod R25)	28	21	4	110
R6 = R2 * R28 (mod R25)	6	2	28	110
R6 = R6 + R0 (mod R25)	6	6	0	100
R7 = 1 / R3 (mod R25)	7	3	20	111
R29 = R6 * R7 (mod R25)	29	6	7	110
Saídas: r, s			**Reg Saídas:** R28, R29	

Figura 52: Estados - operação de geração de assinatura digital

OPERAÇÃO – VALIDAR UMA ASSINATURA DIGITAL				
Entradas: Point P, Point Q, int m, int r, int s				
Saídas: int val				
Parâmetros: Px, Py, Qx, Qy, m, r, s			**Reg Parâmetros:** R26, R27, R0, R1, R2, R3, R4	
Cálculo	**Registradores**			**Código Operação**
	Resultado	**Operador 1**	**Operador 2**	
R5 = 1/ R4 (mod R25)	5	4	20	111
R6 = R2 * R5 (mod R25)	6	2	5	110
R7 = R3 * R5 (mod R25)	7	3	5	110
(R8, R9) = R6 * (R26, R27)	8,9	26,27	6,22	001
(R10, R11) = R7 * (R0, R1)	10,11	0,1	7,22	001
(R8, R9) = (R8, R9) + (R10, R11)	8,9	8,9	10,11	010
R5 = 1 * R8 (mod R25)	5	21	8	110
R28 = R5 – R3(mod R25)	28	5	3	101
Saídas: Val		**Reg Saídas:** R28		

Figura 53: Estados - operação de validação de assinatura digital

Temos, finalmente, um módulo capaz de realizar todas as operações determinadas na especificação do projeto, e podemos então realizar os testes necessários e verificar as formas de onda resultantes em cada operação, a fim de validar o nosso coprocessador e iniciar os trabalhos no módulo de interface hardware-software. Para fins de testes, vamos utilizar novamente os parâmetros usados como exemplo na parte teórica deste relatório, ou seja: P=(1,5), d=7, Q=(24,22), e uma mensagem M=(17,10), ainda com os mesmos parâmetros de curva dos itens anteriores.

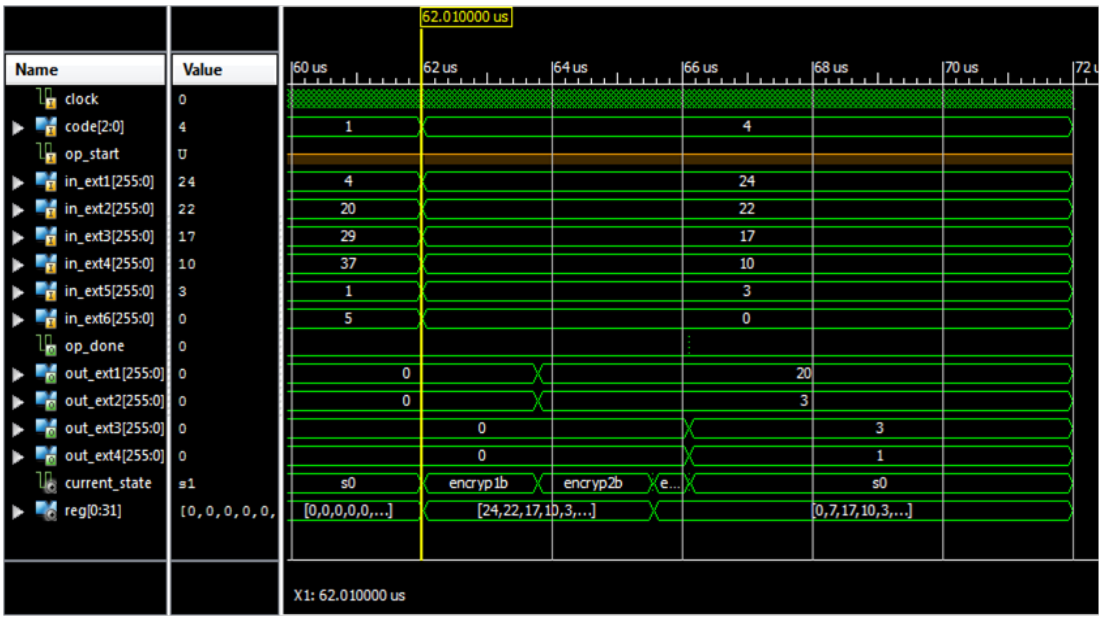


Figura 54: Simulação de uma operação de cifração de uma mensagem

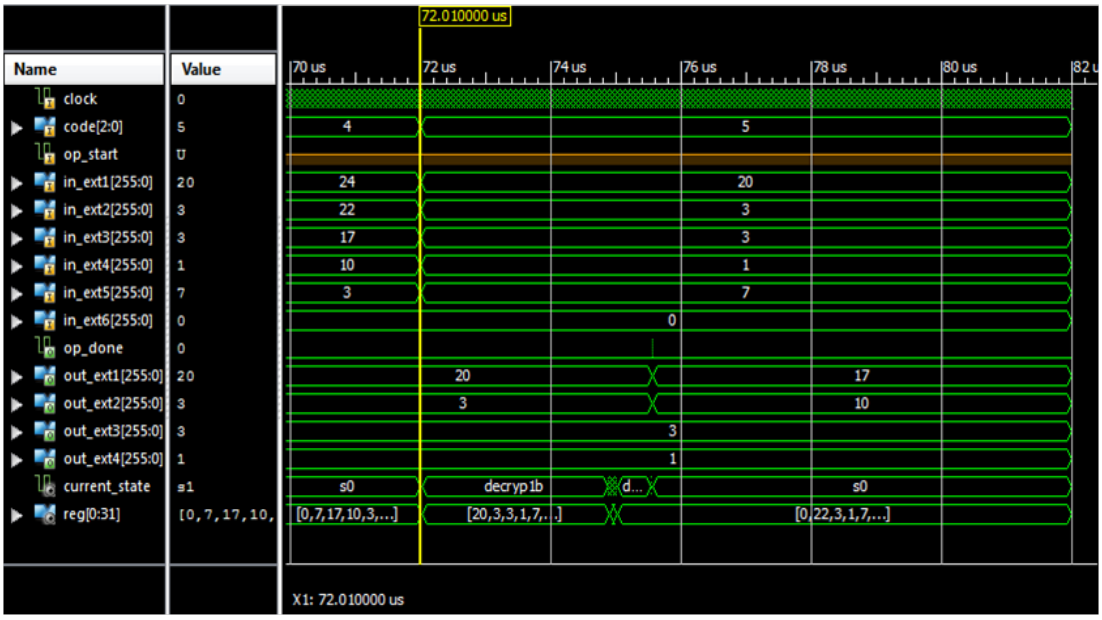


Figura 55: Simulação de uma operação de decifração de uma mensagem

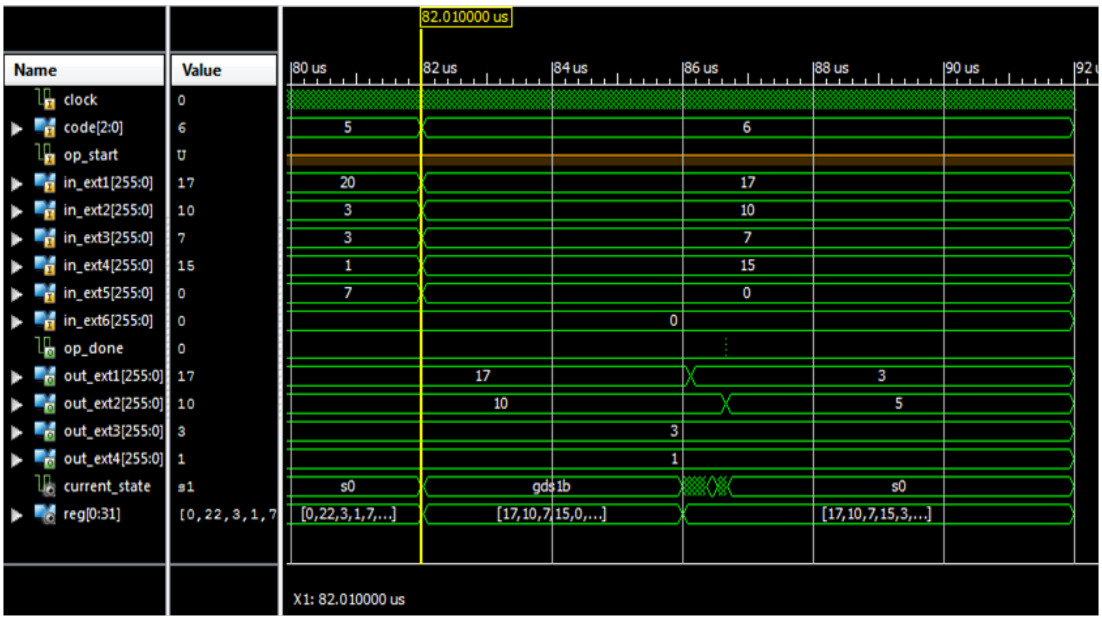


Figura 56: Simulação de uma operação de geração de assinatura digital

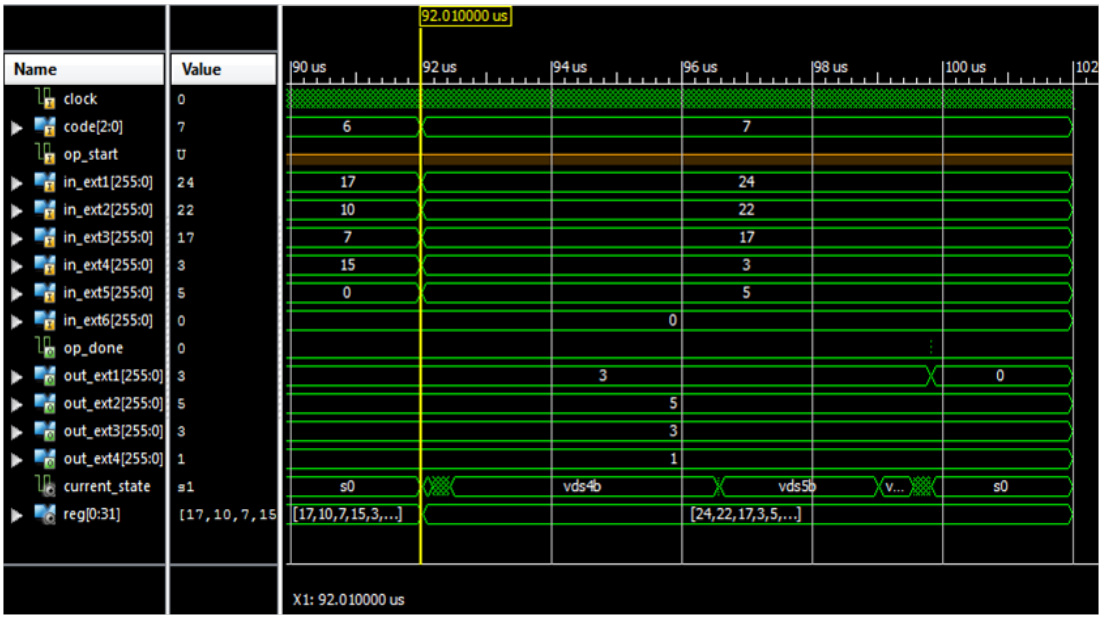


Figura 57: Simulação de uma operação de validação de assinatura digital

8.5 Multiplicador DAR

Ao longo de nosso processo de descrição de hardware e testes com o código, verificamos que, para operandos com uma quantidade muito grande de bits (como é o caso de nosso projeto, que usa 256 bits), surgiam problemas na síntese do multiplicador combinatório. O mapeamento das equações falhava, e mesmo o período mínimo do clock dava indícios de que este circuito não funcionaria bem em frequências mais elevadas.

Para contornar estes problemas, e garantir uma síntese em 256 bits, foi criado um módulo adicional que realiza multiplicação em corpos finitos. Ao contrário do anterior, este trabalha com uma máquina de estados, fazendo uso do algoritmo DAR (Double, Add, Reduce), iterado a cada bit do multiplicador.

Como trata-se de um módulo semelhante ao inversor em corpos finitos, a estrutura da camada de aritmética de corpos finitos foi alterada de modo a acomodar o novo componente.

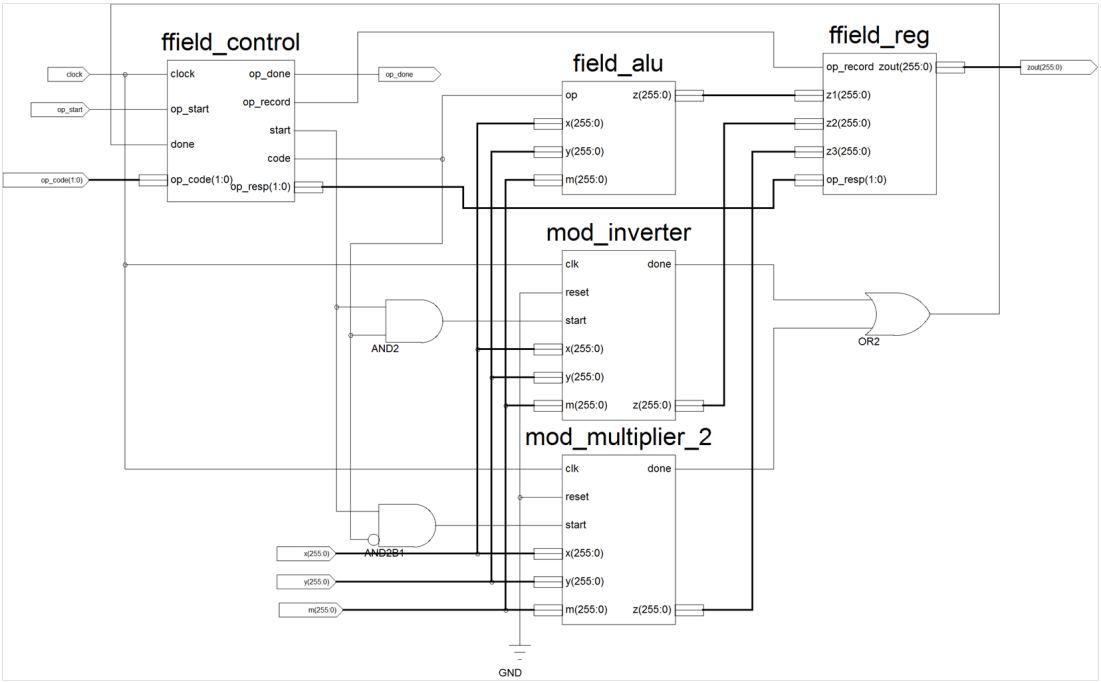


Figura 58: Módulo de aritmética de corpos finitos com o multiplicador DAR

É preciso lembrar, no entanto, que apesar de oferecer os mesmos resultados, esta alternativa significa tempos de respostas maiores para o coprocessador. Nota-se, na nova simulação de multiplicação, que a quantidade de ciclos de clock para terminar a operação aumentou consideravelmente.

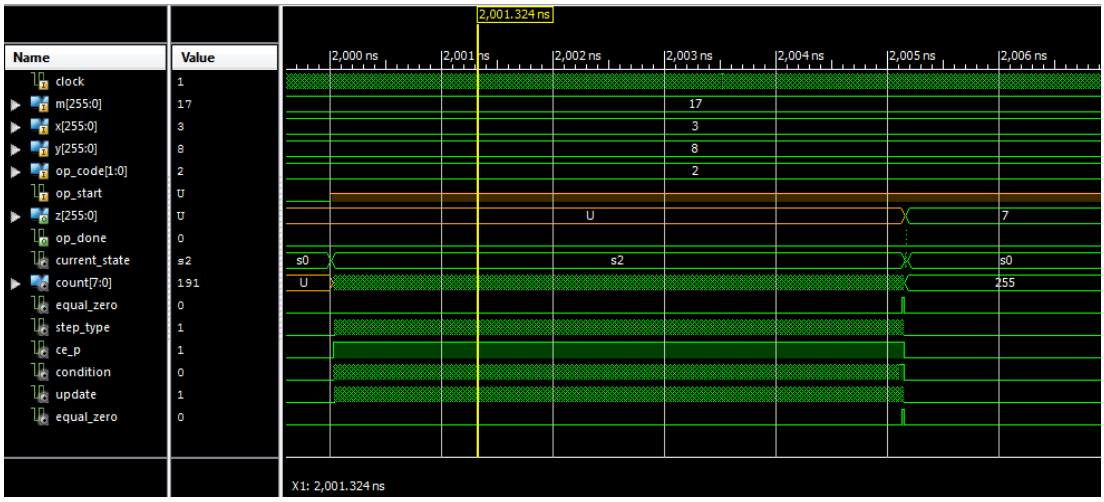


Figura 59: Simulação de multiplicação em corpos finitos com DAR

9 RESULTADOS E TESTES

9.1 Síntese do Coprocessador

Uma vez que já temos uma descrição de hardware pronta e devidamente simulada, utilizamos o Xilinx ISE para sintetizar o coprocessador criptográfico. Segue o relatório final da síntese do hardware projetado (e fazendo uso do multiplicador *DAR*):

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : Coprocessor.ngr
Top Level Output File Name         : Coprocessor
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : No

Design Statistics
# IOs                              : 7

Cell Usage :
# BELS                             : 88845
#      GND                         : 1
```

```
#      INV                         : 296
#      LUT1                        : 1058
#      LUT2                        : 1925
#      LUT2_D                      : 1
#      LUT2_L                      : 6
#      LUT3                        : 30947
#      LUT3_D                      : 19
#      LUT3_L                      : 14
#      LUT4                        : 21651
#      LUT4_D                      : 68
#      LUT4_L                      : 338
#      MUXCY                       : 3270
#      MUXF5                       : 15551
#      MUXF6                       : 6689
#      MUXF7                       : 2576
#      MUXF8                       : 1288
#      VCC                         : 1
#      XORCY                       : 3146
# FlipFlops/Latches                : 14275
#      FD                          : 10507
#      FD_1                        : 1
#      FDC                         : 9
#      FDE                         : 2304
#      FDE_1                       : 8
#      FDPE                        : 1
#      FDR                         : 14
#      FDRE                        : 1312
#      FDS                         : 58
```

```
#      FDSE                          : 61
# Clock Buffers                      : 1
#      BUFGP                         : 1
# IO Buffers                         : 6
#      IBUF                          : 5
#      OBUF                          : 1
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                    33068  out of   8672   381% (*)
 Number of Slice Flip Flops:          14275  out of  17344    82%
 Number of 4 input LUTs:              56323  out of  17344   324% (*)
 Number of IOs:                           7
 Number of bonded IOBs:                   7  out of    250     2%
 Number of GCLKs:                         1  out of     24     4%

WARNING:Xst:1336 -  (*) More than 100% of Device resources are used

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.
```

```
-------------------------
```

```
=========================================================================
```

Nota-se que, apesar de termos um coprocessador sintetizável que atende aos requisitos operacionais deste projeto, o relatório aponta que a FPGA que estamos utilizando não possui recursos suficientes para implementa-lo devidamente, e portanto, uma FPGA mais robusta seria necessária.

Isto ocorre, principalmente, por conta da natureza hierárquica da arquitetura que projetamos. Como cada módulo foi projetado em uma estrutura de controlador, banco de registradores, e unidade aritmética, quando trabalhamos com operandos de grande quantidade de bits, a estrutura acumulativa de registradores, que cresce em tamanho e redundância a medida que nos aproximamos das camadas superiores do coprocessador, consome recursos da FPGA, mais adequada para trabalhar com memórias do tipo RAM.

De modo a conseguir implementar devidamente o nosso hardware com os recursos físicos que temos, é necessário fazer concessões no nível de segurança. Podemos, para isso, reduzir o número de bits dos operandos para 64 bits, o que torna o hardware compatível com a FPGA, porém mais vulnerável a ataques. Segue o relatório:

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : Coprocessor.ngr
Top Level Output File Name         : Coprocessor
Output Format                      : NGC
```

```
Optimization Goal                 : Speed
Keep Hierarchy                    : No

Design Statistics
# IOs                             : 7

Cell Usage :
# BELS                            : 22941
#      GND                        : 1
#      INV                        : 97
#      LUT1                       : 289
#      LUT2                       : 895
#      LUT3                       : 7971
#      LUT3_L                     : 2
#      LUT4                       : 5447
#      LUT4_D                     : 6
#      LUT4_L                     : 24
#      MUXCY                      : 909
#      MUXF5                      : 3827
#      MUXF6                      : 1672
#      MUXF7                      : 644
#      MUXF8                      : 322
#      VCC                        : 1
#      XORCY                      : 834
# FlipFlops/Latches               : 3775
#      FD                         : 760
#      FDC                        : 13
#      FDE                        : 576
```

```
#      FDP                         : 2
#      FDPE                        : 1
#      FDR                         : 15
#      FDRE                        : 352
#      FDS                         : 1932
#      FDS_1                       : 1
#      FDSE                        : 59
#      LD                          : 64
# Clock Buffers                    : 2
#      BUFG                        : 1
#      BUFGP                       : 1
# IO Buffers                       : 6
#      IBUF                        : 5
#      OBUF                        : 1
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s1200efg320-4

 Number of Slices:                     7742  out of   8672    89%
 Number of Slice Flip Flops:           3775  out of  17344    21%
 Number of 4 input LUTs:              14731  out of  17344    84%
 Number of IOs:                           7
 Number of bonded IOBs:                   7  out of    250     2%
 Number of GCLKs:                         2  out of     24     8%
```

```
---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------
```

==========

9.2 Comparação com Software

A última etapa deste projeto é avaliar como o hardware que projetamos se compara às alternativas de software que efetuam as mesmas operações. Para isso, vamos utilizar o nosso terminal escrito em C++, com a biblioteca Cryptopp, para avaliar o tempo médio de realizar as 4 operações necessárias (cifrar e decifrar uma mensagem, gerar e validar a assinatura digital de uma mensagem) sobre os parâmetros iniciais que definimos no início deste relatório. *Processador utilizado: Intel i3 2.1GHz*

Tabela 2: Tempos de Execução - Software com biblioteca Cryptopp

Operação	*Tempo (ms)*
Cifrar uma mensagem	21.809
Decifrar uma mensagem	28.273
Gerar uma assinatura digital	31.722
Validar uma assinatura digital	52.027

Para medir os tempos de execução do coprocessador, torna-se mais flexível lidar diretamente com o número de ciclos de clock. Equacionamos, com base em nosso código e simulações, a quantidade de ciclos de clock que as operações criptográficas gastam em cada módulo do coprocessador:

Tabela 3: Número de ciclos de clock para cada operação - Multiplicador *DAR*

Módulo	*Operação*	*Número de ciclos*
Aritmética de Corpos Finitos	Soma	3
	Subtração	3
	Multiplicação	515
	Inversão	259
Aritmética de Curvas Elipticas	Soma de Pontos	1836
	Dobro de um Ponto	2868
	Soma	7
	Subtração	7
	Multiplicação	519
	Inversão	263
Multiplicação Escalar de Pontos	Multiplicação de Ponto	1087620
	Soma de Pontos	1840
	Dobro de um Ponto	2872
	Soma	11
	Subtração	11
	Multiplicação	523
	Inversão	267
Controlador Principal	Cifrar uma mensagem	2177090
	Decifrar uma mensagem	1089481
	Gerar assinatura digital	1089478
	Validar assinatura digital	2178942

Como a nossa FPGA opera a 50MHz, vamos considerar um período de 20ns para cada ciclo de clock. Sendo assim, os tempos de execução para estas operações em hardware são:

Tabela 4: Tempos de Execução - Coprocessador com multiplicador *DAR*

Operação	*Tempo (ms)*
Cifrar uma mensagem	43.542
Decifrar uma mensagem	21.790
Gerar uma assinatura digital	21.790
Validar uma assinatura digital	43.579

É interessante também avaliar como a unidade de multiplicação combinatória afetaria o desempenho do coprocessador, visto que grande parte das operações criptográficas fazem chamadas de multiplicação em corpos finitos pelo menos uma vez:

Tabela 5: Número de ciclos de clock para cada operação - Multiplicador Combinatório

Módulo	*Operação*	*Número de ciclos*
Aritmética de Corpos Finitos	Soma	3
	Subtração	3
	Multiplicação	3
	Inversão	259
Aritmética de Curvas Elipticas	Soma de Pontos	300
	Dobro de um Ponto	308
	Soma	7
	Subtração	7
	Multiplicação	7
	Inversão	263
Multiplicação Escalar de Pontos	Multiplicação de Ponto	137348
	Soma de Pontos	304
	Dobro de um Ponto	312
	Soma	11
	Subtração	11
	Multiplicação	11
	Inversão	267
Controlador Principal	Cifrar uma mensagem	275010
	Decifrar uma mensagem	137673
	Gerar assinatura digital	137670
	Validar assinatura digital	275326

Tabela 6: Tempos de Execução - Coprocessador com multiplicador combinatório

Operação	*Tempo (ms)*
Cifrar uma mensagem	5.500
Decifrar uma mensagem	2.753
Gerar uma assinatura digital	2.753
Validar uma assinatura digital	5.507

Podemos verificar que o coprocessador com módulo de multiplicação DAR (o mais lento) já apresenta tempos de execução da mesma ordem de magnitude do software, mesmo rodando a 50MHz, contra 2.1GHz do processador convencional. Se considerarmos o coprocessador combinatório, este ainda leva a vantagem de ser na ordem de 10 vezes mais rápido.

Projetistas, então, poderiam escolher entre duas alternativas: manter a frequência mais baixa do coprocessador como forma de economizar bateria e evitar aquecimento, ou igualar as frequências dos dispositivos, e oferecer uma alternativa em hardware consideravelmente mais rápida do que em software.

Tabela 7: Tempos de Execução - Comparação em 2.0 GHz

	Tempo Software (ms)	*Tempo Hardware (ms)*	*Tempo Hardware (ms)*
Operação		*<DAR>*	*<Combinatório>*
Cifrar uma mensagem	21.809	1.089	0.138
Decifrar uma mensagem	28.273	0.545	0.069
Gerar uma assinatura	31.722	0.545	0.069
Validar uma assinatura	52.027	1.089	0,138

10 CONCLUSÕES

Ao longo deste projeto, detalhamos os conceitos teóricos da criptografia de curvas elípticas, fizemos a especificação de um coprocessador criptográfico, e concluímos o seu desenvolvimento. Como forma de avaliar o nosso resultado, precisamos voltar ao início, e responder os questionamentos que fizemos ainda na fase de determinação de objetivos e metodologia:

- O dispositivo faz o que é esperado?

  Sim. O requisito funcional de nosso dispositivo era a realização de 4 operações criptográficas sobre curvas elípticas, e nossas simulações demonstraram que o coprocessador é funcionalmente capaz de realizar todas elas, gerando resultados corretos, e escalável para qualquer tamanho de coeficiente e parâmetro de curva, desde que implementado em hardware que suporte este tamanho.

- O dispositivo cumpre com os requisitos de tempo?

  Sim. Em nossas comparações com alternativas em software, verificamos que o coprocessador atinge resultados semelhantes com frequências muito menores, ou seja, o tempo de execução das operações criptográficas se torna consideravelmente menor do que quando executadas em um processador convencional com a mesma frequência, atendendo, assim, aos requisitos propostos de vazão e tempo de resposta.

- O dispositivo é viável para aplicações móveis?

  De modo a terminar o projeto dentro de seu ciclo de desenvolvimento pré-definido, fizemos uso de uma arquitetura modular, de fácil depuração, e com fluxos de dados simples e independentes. O resultado é uma camada adicional de redundância de hardware, que não é ideal quando estamos lidando com números muito grandes de bits por operando, por consumir mais recursos físicos. Por conta disso, de modo a implementar o coprocessador com uma FPGA, foi necessário fazer algumas alterações no tamanho das variáveis, o que implica em redução no nível de segurança. Entretanto, em placa dedicada e com otimização em seu fluxo de dados, o coprocessador pode atingir melhores resultados nos requisitos de área. Já a possibilidade de operar com tempos de execução satisfatórios, em frequências consideravelmente baixas de clock , torna o coprocessador vantajoso dentro dos requisitos de consumo de energia e calor dissipado.

- O dispositivo é necessário para o sistema?

  Enfim, temos um dispositivo capaz de realizar todas as funções para as quais foi projetado, e operando de forma muito mais eficiente do que as alternativas em software. A avaliação da necessidade do dispositivo em um sistema passa a depender muito de sua utilização: mesmo com tempos de resposta maiores, as alternativas de software ainda se encontram na escala de milissegundos, que podem ser consideradas imediatas ao usuário final. As aplicações que apresentam requisito de alta vazão, entretanto, podem se beneficiar muito de um hardware capaz de realizar um número muito maior de operações criptográficas em um determinado intervalo de tempo. Retomando o caso de uso detalhado no início deste relatório, podemos concluir que a demanda da <loja> por um hardware como este é maior do que a demanda do <comprador>.

REFERÊNCIAS

BROWN, D. R. L. *Standards for Efficient Cryptography 1: Elliptic Curve Cryptography*. [S.l.]: Certicom Corp, 2009.

DESCHAMPS, J.-P. D.; BIOUL, G. J. A.; SUTTER, G. D. *Synthesis of Arithmetic Circuits FPGA, ASIC, and Embedded Systems*. New Jersey: John Wiley and Sons INC, 2006.

DIGILENT. *Nexys 2 Spartan3E FPGA Board*. [S.l.], 2015. Acesso em 24/06/15. Disponível em: <http://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS2>. Acesso em: 18 de dezembro de 2015.

HEKERSON, D.; MENEZES, A.; VANSTONE, S. *Guide to Elliptic Curve Cryptography*. New York: Springer, 2004.

LINKOPINGS UNIVERSITY. *Cryptography Lecture 8 Digital Signatures, Hash Functions*. 2014.

VOGEL, L. *Eclipse IDE Tutorial*. [S.l.], 2014. Acesso em 21/07/15. Disponível em: <http://www.vogella.com/tutorials/Eclipse/article.html>. Acesso em: 18 de dezembro de 2015.

XILINX. *ISE 11 InDepth Tutorial (UG695 v 11.2)*. [S.l.], 2009.

XILINX. *Spartan3E FPGA Family Datasheet*. [S.l.], 2013.