Java Cryptographic Library for Smartphones

C. Okida, D. Goya and R. Terada

***Abstract*— A JME-compliant cryptographic library for mobile application development is introduced in this paper. The library allows cryptographic protocols implementation over elliptic curves with different security levels and offers symmetric and asymmetric bilinear pairings operations, as Tate, Weil, and Ate pairings.**

***Keywords*— Cryptographic Library, Java Micro Edition, Mobile Application, Elliptic Curve Cryptography, Bilinear Pairings.**

I. INTRODUÇÃO

DURANTE o projeto e desenvolvimento de novas aplicações para dispositivos móveis, é recorrente a necessidade de conciliar desempenho com facilidade de portar o código escrito para uma grande variedade de modelos e tipos de dispositivos. Em situações assim, a plataforma Java, mais especificamente a JME, Java Micro Edition [1], é uma opção conveniente para desenvolvedores de software, dada a significativa oferta de *smartphones* compatíveis com a tecnologia. Devido à portabilidade de código da Java, desenvolvedores se desvencilham de preocupações em adaptar um determinado programa às mais variadas configurações e diretivas de otimização em compiladores que cada fabricante de dispositivo móvel impõe.

Quando a aplicação exige o uso de mecanismos de criptografia ou a codificação de protocolos criptográficos, há poucas ferramentas disponíveis que sejam compatíveis com JME. Dentre essas, as APIs criptográficas do projeto Bouncy Castle [2] são as mais conhecidas. A biblioteca Bouncy Castle oferece recursos principalmente para criptografia simétrica, incluindo diversificadas cifras de bloco, modos de operação e funções de hash, além de MAC e cifra de fluxo. Para criptografia assimétrica, de chave pública, a Bouncy Castle se resume essencialmente ao RSA, em várias implementações padronizadas, acordo de chaves Diffie-Hellman e alguns esquemas de assinatura digital. Também são fornecidas implementações sobre curvas elípticas [3][4] para a assinatura digital ECDSA e Diffie-Hellman.

No entanto, a Bouncy Castle apresenta-se limitada quando se faz necessária a implementação de novos protocolos criptográficos assimétricos, que surgiram nos últimos dez ou quinze anos. Mais especificamente, notam-se restrições para a seleção de curvas elípticas e inexiste a opção de codificação de protocolos baseados em emparelhamentos bilineares [5][6][7], que têm possibilitado a construção de novas primitivas criptográficas, como encriptação baseada em identidade [8], busca em dados cifrados [9], chave pública sem certificado [10], entre outras aplicações.

Portanto, julga-se interessante que desenvolvedores de software tenham à disposição uma biblioteca criptográfica, distribuída como software livre, com recursos ampliados para geração de curvas elípticas seguras e com interface para a codificação de protocolos baseados em emparelhamentos.

Nossas Contribuições

Neste artigo, apresentamos uma biblioteca codificada em Java, com compatibilidade com JME, que permite a construção de aplicações criptográficas com maior flexibilidade para escolha de nível de segurança, criação de curvas elípticas e operações de emparelhamentos bilineares. A biblioteca é distribuída como software livre e está em testes com uma aplicação de telessaúde, que envolve transmissão de prontuários médicos entre telefones celulares e servidor.

Organização do Trabalho

Antes de apresentarmos nossa proposta, descrevemos brevemente as bibliotecas existentes com recursos para criptografia de curvas elíptica, que são codificadas em Java, na Seção II. Na sequência, na Seção III, descrevemos a arquitetura e implementação de nossa biblioteca, seguida de exemplos de uso. Na Seção IV, mostramos resultados experimentais efetuados em dispositivo móvel. Por fim, apresentamos conclusões e trabalhos futuros.

II. BIBLIOTECAS CRIPTOGRÁFICAS EM JAVA

As principais bibliotecas criptográficas em Java com suporte a curvas elípticas são apresentadas na Tabela I.

Entre esses projetos, apenas dois, Bouncy Castle e o Secure SMS podem ser usados em dispositivos móveis, por isso vamos detalhar cada um deles a seguir.

A. Bouncy Castle

O projeto Bouncy Castle visa à construção de uma biblioteca criptográfica, que atenda a maior quantidade de curvas padronizadas pelo NIST. Possui API de criptografia para Java e C#, além de possuir uma versão em JME, para dispositivos móveis. A biblioteca implementa curvas elípticas primas e binárias, padronizadas pelo NIST, está sob a licença MIT e, por ser australiana, as restrições americanas de exportação de criptografia de software não se aplicam a ela.

TABELA I
BIBLIOTECAS CRIPTOGRÁFICAS EM JAVA.

Biblioteca	Mantenedor
Bouncy Castle	The Legion of the Bouncy Castle [2]
FlexiProvider	Darmstadt University of Technology [11]
Java Crypto API	Oracle America, Inc. [12]
jPBC	Dip. Informatica - Universitá Salerno [13]
Secure SMS	LARC – USP [14]

C. Okida, Universidade de São Paulo (USP), São Paulo, Brasil, cleberok@ime.usp.br
D. Goya, Universidade de São Paulo (USP), São Paulo, Brasil, dhgoya@ime.usp.br
R. Terada, Universidade de São Paulo (USP), São Paulo, Brasil, rt@ime.usp.br

As curvas elípticas implementadas alcançam nível de segurança de até 128 bits, conforme o documento SP-800-57a do NIST, de 2007. Essa biblioteca não oferece suporte para emparelhamentos bilineares.

B. Secure SMS

O projeto *Secure SMS* [15] desenvolve serviços de segurança para a troca de mensagens SMS (*Short Message Service*) entre aparelhos de telefonia celular. Sua implementação foi desenvolvida por Cruz, Pereira, Silva e Barreto, sendo este último o autor do núcleo do código criptográfico que inclui operações sobre curvas elípticas e cálculo de emparelhamento. Por ser uma aplicação, o *Secure SMS* não é distribuído como uma biblioteca criptográfica. É necessário obter o código completo do sistema e realizar, posteriormente, a separação das funções criptográficas para reusá-las em outros projetos.

A aplicação está toda sob licença GPL v.3, que implica ser um software permanentemente aberto.

A implementação inclui curvas elípticas do tipo ordinárias, mais especificamente as curvas MNT [16], com nível de segurança de até 313 bits. O emparelhamento bilinear oferecido é do tipo assimétrico Ate com nível de segurança de até 112 bits. Não há opção de curvas elípticas supersingulares nem emparelhamento simétrico.

III. Biblioteca jMobilePBC

Motivados pela necessidade de uma biblioteca com recursos para implementação de um mecanismo para prover sigilo na comunicação entre um dispositivo móvel e um servidor de banco de dados, além da insuficiência das alternativas disponíveis até então, codificamos uma nova biblioteca, nomeada jMobilePBC. Em especial, precisávamos oferecer soluções para dispositivos móveis com intervalo de nível de segurança mais amplo (ver Tabela II) e de maior flexibilidade para escolha de curvas e emparelhamentos bilineares simétrico e assimétrico.

TABELA II
CAPACIDADE E NÍVEL DE SEGURANÇA (BITS).

Biblioteca	Tamanho Máximo p/Inteiros	Nível de Segurança			
		Curva Elíptica	Emparelhamento		
			Ate	Weil	Tate
Bouncy Castle	521	256	–	–	–
B-mnt4	463	313	112	–	–
jMobilePBC	1024	512	128	128	128

A jMobilePBC implementa curvas elípticas sobre corpos finitos primos. Ela permite que protocolos baseados em emparelhamentos que utilizam emparelhamentos tanto do tipo simétrico (o Weil e Tate, por exemplo) como do tipo assimétrico (por exemplo, Ate) sejam implementados.

Mesmo que a preocupação inicial estava centrada na segurança de dispositivos móveis, era necessário que a biblioteca pudesse ser executada no ambiente servidor. Por esse motivo, ela também se encontra disponível para utilização na versão J2SE.

A biblioteca está sob licença BSD, o que significa que é um software livre que pode ser modificado e distribuído sem restrição, a menos da parte que envolve emparelhamento `Ate`. Isto é, as classes `Curva2`, `Ponto2`, `Fp12` e a própria `Ate`, que são parte integrante do projeto *Secure SMS*, que por sua vez é licenciado sob GPL v.3, precisam estar permanentemente abertas.

A jMobilePBC encontra-se disponível em [17]. A distribuição inclui a versão para JME das classes `BigInteger` e `SecureRandom`, permitindo a execução da biblioteca em dispositivos móveis.

A origem da jMobilePBC, está relacionada com o projeto Borboleta, disponível em [18]. Esse projeto se vale de dispositivos móveis para auxiliar os profissionais da saúde nos atendimentos domiciliares, como uma ferramenta de apoio à coleta de informações para um Sistema Móvel de Prontuário Eletrônico, que está fortemente baseado no conceito de acompanhamento da situação clínica dos pacientes.

Como requisito primordial do sistema, os dados de prontuários médicos devem trafegar de forma segura entre dispositivos móveis e servidor do sistema de saúde, com garantia de autenticação de ambos os lados, além de sigilo em elevado nível de segurança e integridade dos dados em trânsito. Como os PDAs possuem poder de processamento relativamente baixo e conexões de pequena largura de banda, o sistema exige algoritmos otimizados para viabilizar um bom desempenho.

A. Arquitetura

Vamos descrever a biblioteca jMobilePBC em camadas, conforme ilustrado na Fig. 1. Os níveis inferiores da pirâmide representam as camadas que possuem menor complexidade; objetos nos níveis superiores usam objetos das camadas mais baixas.

Figura 1. Classes principais em camadas.

Uma breve descrição de cada camada é dada a seguir:

- Camada 1. Operações com precisão arbitrária: inclui aritmética sobre números inteiros de tamanho arbitrário; nesta camada está localizada a classe `BigInteger` e `SecureRandom`, codificado para a plataforma JME;
- Camada 2. Operações sobre corpos finitos: definem-se as operações algébricas sobre corpos finitos de característica prima;

- Camada 3. Curvas elípticas: definem-se as curvas elípticas supersingulares e ordinárias e suas operações;
- Camada 4. Emparelhamentos bilineares: cálculo dos emparelhamentos de Tate, Weil e Ate;
- Camada 5. Aplicações: é nessa camada onde são implementadas as aplicações; como exemplos são fornecidos duas codificações de protocolos de acordo de chaves.

Passamos a descrever como foram implementadas as camadas 1 a 4.

B. Precisão Arbitrária

Até a versão 2.5.2 da WTK para JME, não há implementação de uma classe para números inteiros de tamanho arbitrário. Por esse motivo, se fez necessária a obtenção de um gerenciador eficiente para esse tipo de estrutura.

Dentre as operações sobre números inteiros, a mais crítica é a multiplicação. Desse modo, foi implementado o algoritmo de Karatsuba [19] otimizado para sistemas de 32 bits de tamanho da palavra.

A BigInteger aceita números de tamanho no mínimo 56 bits até 1024 bits, atentando ao acréscimo de 8 bits no intervalo.

A classe `SecureRandom` foi criada com o intuito de melhorar a eficiência dos sorteios dos valores aleatório.

C. Corpos Finitos

Da mesma forma que no algoritmo RSA, os cálculos envolvidos em curvas elípticas necessitam da estrutura matemática de corpos finitos, representado como F_p, onde p é a característica do corpo.

Essa estrutura de corpo finito, assim como um grupo, permite que trabalhemos a abstração de pontos sobre uma curva elíptica, como sendo uma estrutura de conjunto.

A partir dessa ideia, é possível tratar cada elemento do corpo finito com seu respectivo conjunto, o que nos leva às estruturas das extensões desses corpos, representadas pelo expoente de cada corpo, como a extensão de grau *2* para o caso F_p^2 por exemplo.

Na Fig. 2, apresentamos a relação entre um elemento algébrico e o conjunto do qual faz parte, como um corpo finito e suas extensões.

Os conjuntos são descritos a seguir:

- F_p: representa o corpo finito de característica prima, onde p é um primo ou uma potência de primo; a representação desse conjunto é um número inteiro;
- F_p^2: caracteriza a extensão de grau *2* do corpo finito F_p; sua representação é dada como um número complexo com parte real e imaginária, para tirar proveito da propriedade da multiplicação complexa;
- F_p^{12}: representa a extensão de grau *12* do corpo finito, que consiste em *6* elementos de F_p^2.

A classe `Fp` é usada para definir os elementos das curvas supersingulares (descritas na seção a seguir) e as extensões `Fp2` e `Fp12` são usadas para definir os elementos das curvas ordinárias do tipo BN [20].

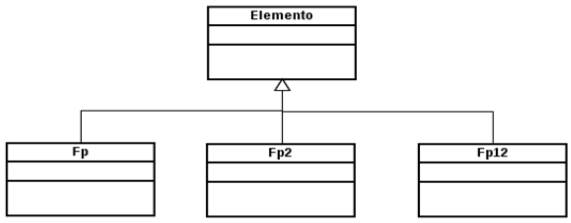

Figura 2. Classes principais da camada 2, para corpos finitos.

Não implementamos corpos de característica binária porque, apesar de serem eficientes para serem operadas, são mais rápidas para serem atacadas, quando comparadas com as de característica prima [21], além de viabilizarem apenas curvas supersingulares, conforme discutimos na próxima seção.

Calculamos os elementos por meio da aritmética dos corpos finitos, mas a representação de fato será abordada nas próximas seções.

D. Curvas Elípticas

Podemos definir os corpos finitos como subconjuntos de pontos pertencentes a uma curva elíptica.

As curvas elípticas podem ser de dois tipos:

- Curvas subjacentes: são as curvas definidas sobre um corpo finito, e são da forma $E(F_p)$: $y^2 = x^3 + ax + b$, onde a, b estão em Z_p. Tais curvas estão definidas na classe `Curva` (Fig. 3) e são necessárias para todo tipo de aplicação envolvendo curvas elípticas;
- Curvas torcidas: são as curvas definidas sobre uma extensão de um corpo finito, e são da forma forma $E(F_p^n)$: $y^2 = x^3 + ax + b$, onde a, b estão em F_p^n e n é o grau da extensão do corpo, tais curvas estão definidas na classe `Curva2` e são necessárias quando se utilizam emparelhamentos bilineares assimétricos.

As aplicações criptográficas tem se apoiado na dificuldade do problema do logaritmo discreto para esconder segredos. Essa dificuldade está relacionada ao desconhecimento de algoritmos eficientes que calculam o valor de uma chave secreta, digamos a, a partir de um dado ponto gerador P de um grupo G, e dado um ponto aP de G, onde a é um valor aleatório escolhido uniformemente ao acaso de Z_p.

Retomando a abstração de grupo como um conjunto, e devido à característica discreta do sistema, parâmetros públicos e secretos são definidos aos pares como *(aP,a)*, com aP sendo um ponto gerado por a e P.

Esses pontos são associados à curva, então o ponto definido na classe `Ponto` pertence à `Curva`, da mesma forma que um objeto de `Ponto2` é elemento de uma curva pertencente à classe `Curva2`. Isto é, o `Ponto` possui coordenadas no corpo dos inteiros Z_p, e o `Ponto2` possui coordenadas na extensão de grau dois de F_p, como apresentamos na Fig. 3.

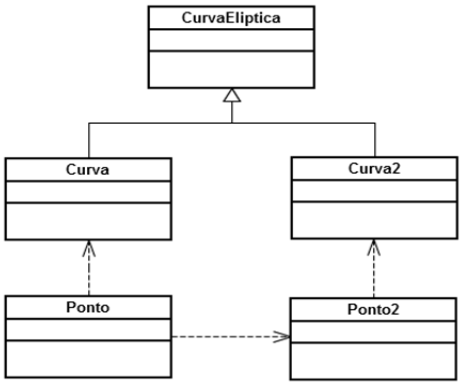

Figura 3. Classes principais da camada 3, para curvas elípticas.

A biblioteca permite, ao contrário das demais, flexibilidade ao implementar tipos diferentes de curvas subjacentes, sejam elas supersingulares ou ordinárias. Ambas podem ser implementadas com a classe `Curva` da jMobilePBC, por estarem definidas sobre corpos de característica prima.

As curvas supersingulares são as curvas mais simples existentes, definidas sobre corpos de característica prima, binária ou ternária com grau de mergulho $k \leq 6$ [22] e são aplicadas no emparelhamento de Weil e Tate, como descrito na próxima seção.

A alternativa para as curvas supersingulares são as famílias especiais de curvas ordinárias, também chamadas de não-supersingulares.

Neste caso, o emparelhamento (Tate, por exemplo) recai sobre o corpo estendido F_p^k, onde k é o grau de mergulho da curva.

Um exemplo de curvas ordinárias são as curvas BN, que constituem uma das classes mais versáteis de curvas elípticas amigáveis a emparelhamentos. Elas são conhecidas por facilitar o desenvolvimento de emparelhamentos para o nível de segurança de 128 bits.

Nossa implementação contempla as curvas supersingulares de grau de mergulho *2* e curvas BN de grau de mergulho de grau *12*, isto é, a classe `Curva` faz uso da classe `Fp2` definida na camada de corpos finitos, enquanto a classe `Curva2` usa as classes `Fp2` e `Fp12`.

Na próxima sessão, apresentamos os conceitos de emparelhamento bilinear e os tipos aqui citados.

E. Emparelhamentos Bilineares

Um emparelhamento bilinear é uma função definida sobre curvas elípticas, onde há grupos cíclicos G_1, G_2 e G_T de ordem prima q. Um emparelhamento bilinear admissível [8], é um mapeamento $e : G_1 \times G_2 \rightarrow G_T$, que satisfaz as seguintes condições:

1. Bilinearidade: para todo P em G_1, Q em G_2 e a,b em Z_p, $e(aP,bQ) = e(P,Q)^{ab} = e(abP,Q) = e(P,abQ)$
2. Não Degeneração: não leva todos os pares $G_1 \times G_2$ à identidade em G_T
3. Computabilidade: existe algoritmo eficiente que calcula $e(P,Q)$ para todo P em G_1 e Q em G_2

Os emparelhamentos bilineares podem ser de dois tipos:

- Simétricos: onde $G_1 = G_2$, consequentemente temos a propriedade de $e(P, Q) = e(Q, P)$;
- Assimétricos: onde $G_1 \neq G_2$.

A Fig. 4 apresenta as classes dos emparelhamentos implementados, que são os de Weil, Tate e Ate.

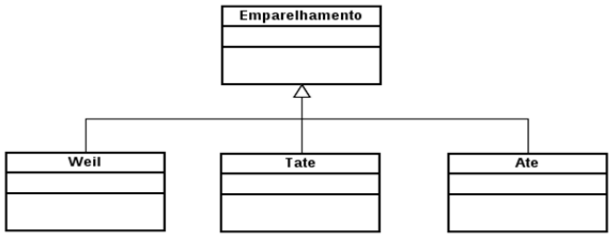

Figura 4. Classes principais da camada 4, para emparelhamentos.

- `Weil` implementa o emparelhamento de Weil para curvas supersingulasres. O resultado deste emparelhamento é obtido a partir de duas iterações do emparelhamento de Tate;
- `Tate` implementa o emparelhamento de Tate para curvas supersingulares. Este emparelhamento é obtido a partir de dois pontos P e Q de G_1, mapeado para um grupo G_T, que é um `Elemento` de F_p^2 para a curva implementada;
- `Ate` implementa o emparelhamento Ate, que é uma generalização do emparelhamento de Tate, mas para curvas ordinárias.

Outros detalhes sobre implementação e aspectos teóricos são descritos em [23].

F. Exemplo de Código-Fonte em Java

Vamos mostrar como utilizar os principais métodos da biblioteca para criação de aplicações.

O algoritmo de multiplicação de Karatsuba da classe `BigInteger` foi implementado para ser mais um método dessa classe; é invocado por `mult()`.

Para gerar um valor aleatório s de Z_p, por exemplo, prosseguimos como apresentado no código da Fig. 5.

```
// criar uma semente
byte[] randSeed = new byte[20];
(new SecureRandom()).nextBytes(randSeed);
// criar o objeto aleatório
SecureRandom rnd = new SecureRandom(randSeed);
// cria um inteiro aleatório
// de tamanho (int)numBits
BigInteger s = new BigInteger(numBits, rnd);
```

Figura 5. Geração de um inteiro aleatório.

A classe `Elemento` atua como um "invólucro" em torno da classe `BigInteger`. Isso reduz a dependência da classe `BigInteger` e permite uma alternativa à classe de inteiros de precisão arbitrária para ser usada, ao mesmo tempo que diminui o impacto sobre o resto do código.

Assim, as operações da classe `Elemento` são definidas como na classe `BigInteger`, `add()`, `neg()`, `mult()`, `div()` e `inv()`.

No trecho de código da Fig. 6, apresentamos um exemplo de criação de um corpo finito de característica prima *11* e a operação de soma de dois elementos deste corpo.

```
// criar um corpo
Corpo f11 = new Fp(new BigInteger("11"));
// criar os elementos do corpo finito
Elemento a = f11.elemento(new BigInteger("4"));
Elemento b = f11.elemento(new BigInteger("7"));
// executar a operação de adição pelo objeto Fp
Elemento resultado = f11.add(a, b);
// uma alternativa é executar
//a operação diretamente por
Elemento resultado2 = a.add(b);
```

Figura 6. Operações sobre corpos finitos.

De forma análoga, pode-se criar elementos para corpos de extensão F_p^2 e F_p^{12}.

Se utilizar a mistura de tipos de elementos, por exemplo com a em F_p e b em F_p^2, como pode acontecer nos emparelhamentos, então a super classe é promovida automaticamente a subclasse pela API, de a em F_p resulta em $a+0i$ em a em F_p^2, desde que b pertença a em F_p.

Como as curvas elípticas estão intimamente ligadas ao tipo de corpo que as define, se uma curva $E : y^2 = x^3 + ax + b$ estiver sobre o corpo F_p, então os seus coeficientes a,b também estarão sobre F_p, como no caso da curva subjacente.

Um exemplo de uso da classe `Curva` é apresentado na Fig. 7.

```
// nível de segurança
int nivel = 128;
// criar um corpo finito primo
//com nível de segurança desejado
Corpo fp = new Fp(nivel);
// criar a curva sobre o corpo finito
Curva E = new Curva(fp);
```

Figura 7. Curva subjacente ao corpo finito.

Para as curvas torcidas, os coeficientes estarão definidos sobre o corpo F_p^2, dessa forma a e b são da forma $a = (x_a, y_a)$ e $b = (x_b, y_b)$, onde $j = x_j+iy_j$ e i é a constante imaginária $i = \sqrt{-1}$.

As curvas torcidas estão sempre associadas a uma curva subjacente, pois necessita do corpo base definido na curva subjacente. Um exemplo de implementação é apresentado no código da Fig. 8.

```
// cria a curva torcida associada
//à curva subjacente
Curva2 E2 = new Curva2(E);
```

Figura 8. Implementação de curvas torcidas.

Pela ideia apresentada, que sugere os pontos como elementos pertencentes à curva elíptica, a classe `Ponto` está para `Curva`, assim como `Ponto2` está para `Curva2`, como apresentado no algoritmo da Fig. 9.

```
// cria um ponto P(x1, y1) da curva E
Ponto P = new Ponto(E, x1, y1);
// ponto gerador do grupo
Ponto G = new Ponto(E.G);
// operação de adição de dois pontos P+G
Ponto R = new Ponto(P.add(G));
// cria um ponto Q(x2, y2) da curva E2
Ponto2 Q = new Ponto2(E2, x2, y2);
```

Figura 9. Operações sobre pontos de curvas torcidas.

Tais classes possuem outras opções de construtor, devido à variedade de estruturas utilizadas, como, por exemplo, a associação de `Elemento` com os tipos de corpos F_p.

As funções emparelhamentos mapeiam dois pontos P e Q, de seus respectivos corpos finitos, para um `Elemento` do contradomínio da função. Por exemplo, o emparelhamento Ate mapeia P em F_p e Q em F_p^2 para um elemento do corpo F_p^{12}.

O algoritmo da Fig. 10 apresenta um exemplo de implementação do emparelhamento de Tate.

```
// nível de segurança
int nivel = 128;
// cria os corpos finitos
//com nível de segurança desejado
Corpo fp = new Fp(nivel);
Corpo fp2 = new Fp2(nivel);
// criar a curva sobre o corpo finito
Curva E = new Curva(fp);
// cria um ponto P(x1, y1) da curva E
Ponto P = new Ponto(E, x1, y1);
// cria um ponto Q(x2, y2)
Ponto Q = new Ponto(E, x2, y2);
// cria o emparelhamento de Tate
//sobre a curva subjacente
Emparelhamento t = new Tate(E);
// calcula o emparelhamento de Tate
//sobre os pontos P e Q
Elemento resultado = fp2.elemento(
        t.emparelhamentoTate(P, Q));
```

Figura 10. Emparelhamento Tate.

O emparelhamento de Ate é do tipo assimétrico, muito útil em protocolos criptográficos baseados em emparelhamento.

Um exemplo de uso do emparelhamento Ate é apresentado na Fig. 11.

Tais funções são ferramentas elementares para construção de protocolos criptográficos para serem usados nas comunicações das aplicações.

Para exemplificar como usar as camadas inferiores, implementamos na camada de Aplicações dois protocolos de acordo de chave: o MQV [24] e SOK [5].

Protocolos de acordo de chaves são usados para se estabelecer chaves secretas de sessão que, em geral, são usadas como chave secreta em algoritmos simétricos, como cifras de bloco.

```
// cria os corpos finitos
//com nível de segurança desejado
Corpo fp12 = new Fp12(nivel);

// cria a curva torcida associada
//à curva subjacente
Curva2 E2 = new Curva2(E);
// cria um ponto Q(x2, y2) da curva E2
Ponto2 Q = new Ponto2(E2, x2, y2);
// cria o emparelhamento de Ate
//sobre a curva subjacente
Emparelhamento a = new Ate(E, E2);
// executar o emparelhamento de Ate
//nos pontos P e Q
Elemento resultado = fp12.elemento(
            a.emparelhamentoAte(Q, P));
```

Figura 11. Emparelhamento Ate.

O primeiro protocolo, MQV, é um protocolo de acordo de chave com autenticação mútua, padronizado por ANSI X9.63, IEEE 1363-2000, ISO/IEC IS 15946-3, entre outros, e é implementado sobre curvas elípticas; o segundo, SOK, usa emparelhamentos bilineares. Os protocolos MQV e SOK são ambos invocados como apresentado no trecho de código da Fig. 12.

```
// cria um elemento para conter
//a chave secreta do acordo
Elemento sk = new Elemento(null);
// chave secreta é obtida do acordo MQV
sk = MQV(nivel);
// ou
// chave secreta é obtida do
//acordo baseado em emparelhamento SOK
sk = SOK(nivel);
```

Figura 12. Exemplo da utilização de protocolos.

Dessa forma, essas implementações podem ser reutilizadas para aplicações de comunicação e transferência de dados para dispositivos móveis.

IV. Experimentos

Nesta seção, apresentamos alguns resultados experimentais, efetuados em ambiente real sobre um *smartphone*.

Os testes realizados servem para avaliar o desempenho computacional da biblioteca, comparativamente às outras implementações: Bouncy Castle e o núcleo criptográfico do *Secure SMS*.

Foram tomadas medidas de execução das principais operações existentes em cada uma das cinco camadas, descritas na Seção III, sobre a arquitetura da jMobilePBC.

Nas Tabelas III a VII, são apresentados os tempos médios em milissegundos, obtidos sobre o seguinte ambiente de testes:

- Computador

  Acer Aspire 5610Z com processador Intel® T2080 1,73GHz e 1,0GB de memória RAM e sistema operacional Ubuntu 9.04
- Celular

  HTC P3451 com processador OMAP™ 850 201MHz e 128MB de memória RAM e sistema operacional Windows Mobile® 6 Professional
- Compilador

  J2ME Wireless Toolkit 2.5.2 (dispositivo móvel) e J2SE 1.6.0 (computador)

  IDE NetBeans 6.7.1

Nas Tabelas III a IV, denotamos por "B-mnt4" a implementação de Barreto para curvas MNT4 [16] em Java, contida na aplicação *Secure SMS* [15].

Para cada biblioteca, as curvas consideradas foram:

- Bouncy Castle.: B-163 para o nível de 80 bits e P-192 para o nível de 96 bits. Essas curvas são padronizadas pela NIST;
- B-mnt4: E_1/F_p: $y^2 = x^3 - 3x + b$;
- jMobilePBC:
  - Curva supersingular: E_2/F_p: $y^2 = x^3 + 1$;
  - Curva ordinária: E_3/F_p: $y^2 = x^3 + 3$.

A. Metodologia

Para as bibliotecas concorrentes, nós preservamos a integridade do código original e invocamos os métodos à medida que cada objeto fosse necessário, evitando as interfaces que poderiam aumentar o tempo gasto comprometendo a honestidade dos resultados. Sabendo-se que quando um método é invocado por uma interface, este pode ocupar espaço na memória com outros métodos da interface desnecessários para o teste, aumentando o tempo de execução.

Além disso, como os experimentos foram executados no *smartphone*, por falta de mecanismos de *profile* para esta plataforma, houve a preocupação de retirar o maior número de processos que poderiam ser concorrentes à aplicação.

Os experimentos foram construídos para permitir fácil adaptação ao nível de segurança desejado, seguindo o exemplo da biblioteca SMS.

Todos os resultados foram obtidos a partir da média aritmética de 100 execuções de cada teste.

B. Resultados e Análise

A Tabela III apresenta resultados relativos aos testes sobre operações nas camadas 1 e 2. Da camada 1, selecionamos a operação de multiplicação com precisão arbitrária e a denotamos por "x". Para a camada 2, apresentamos os testes para multiplicação (*) e inversão (inv) de elementos sobre o corpo finito criado. Nessa categoria, os testes foram realizados para dois tamanhos: elementos com 160 bits e 256 bits.

TABELA III
TEMPOS PARA OPERAÇÕES SOBRE AS CAMADAS 1 E 2 (MS).

Tamanho →	160 bits			256 bits		
Operações →	x	*	inv	x	*	inv
Bouncy Castle	0,903	1,114	1,103	1,303	1,512	1,504
B-mnt4	0,904	1,113	1,104	1,304	1,513	1,503
jMobilePBC	0,879	0,969	0,985	1,257	1,518	1,498

A jMobilePBC obteve desempenho similar às demais opções, com exceção das operações sobre corpos finitos sobre elementos de 160 bits, nas quais houve um ganho relativo de desempenho de cerca de 13,0% para multiplicação e 10,7% para inversão.

Sobre curvas elípticas, foram testadas as operações de soma de dois pontos distintos (+), soma de dois pontos iguais ou duplicação de um ponto (2P), e multiplicação de um ponto por um escalar (*). Na Tabela IV são apresentados esses testes, sobre curvas com níveis de segurança de 80 bits e 128 bits (isto é, sobre curvas definidas sobre corpos base de 160 e 256 bits, respectivamente). Nesses testes, vemos que a Bouncy Castle apresentou os piores tempos. No nível de segurança de 80 bits, nossa implementação foi cerca de 24,1% mais veloz para soma e multiplicação escalar e cerca de 25,1% mais veloz para duplicação de ponto, relativamente à Bouncy Castle. Já no nível de segurança de 128 bits, nossa implementação foi cerca de 3,9% mais veloz para soma e até 6,6% mais veloz para duplicação de ponto e multiplicação escalar, também relativamente à Bouncy Castle.

TABELA IV
TEMPOS PARA OPERAÇÕES SOBRE CURVAS ELÍPTICAS (MS).

Segurança →	80 bits			128 bits		
Operações →	+	2P	*	+	2P	*
Bouncy Castle	123,2	128,9	153,8	164,3	174,5	187,2
B-mnt4	95,5	98,4	119,2	158,4	163,4	176,5
jMobilePBC	93,6	96,6	116,6	157,9	162,9	175,9

Para os testes sobre cálculo de emparelhamentos, fixamos os níveis de segurança de 80 e 96 bits. A Bouncy Castle não realiza nenhum desses tipos de cálculos, por esse motivo aparece na Tabela V com a sinalização "—" indicando que não se aplica tal medição.

A B-mnt4 implenta apenas o emparelhamento de Ate sobre curvas ordinárias MNT4.

TABELA V
TEMPOS PARA OPERAÇÕES SOBRE EMPARELHAMENTOS (MS).

Seguran. →	80 bits			96 bits		
Emparel. →	Weil	Tate	Ate	Weil	Tate	Ate
Bouncy C.	—	—	—	—	—	—
B-mnt4	—	—	446,2	—	—	521,8
jMobilePBC	11942,7	9744,3	387,0	16745,5	13539,4	461,4

Para testar a camada de Aplicações, usamos os protocolos de acordo de chave MQV e SOK. Os resultados obtidos encontram-se na tabela VI.

Na tabela VII, os tempos para o protocolo SOK se referem a testes com emparelhamento do tipo Ate. Nesse caso, a jMobilePBC realiza os cálculos sobre curvas ordinárias do tipo BN, enquanto B-mnt4 o faz sobre curvas MNT4.

A implementação existente em B-mnt4 não permitiu calcular emparelhamento de Ate com nível de segurança de 128 bits, pois essa configuração da curva MNT não apresenta um método eficiente para alcançar o mesmo nível de segurança.

A jMobilePBC foi cerca de 13,9% mais rápida que B-mnt4 para calcular uma chave secreta com SOK, no nível de 80 bits de segurança, e cerca de 11,5% mais veloz para o nível de 96 bits. Com segurança de 128 bits, o tempo obtido por jMobilePBC para calcular SOK foi de cerca de 567,95 ms, o que é relativamente bom para um celular.

TABELA VI
TEMPOS PARA APLICAÇÕES COM CURVA ELÍPTICA: MQV (MS).

Segurança →	80	96	112	128
Bouncy C.	474,7	671,5	819,0	943,2
B-mnt4	465,7	631,7	757,6	885,5
jMobilePBC	413,2	572,7	689,3	804,5

TABELA VII
TEMPOS PARA APLICAÇÕES COM EMPARELHAMENTO: SOK COM ATE (MS).

Segurança →	80	96	112	128
Bouncy C.	—	—	—	—
B-mnt4	461,87	539,73	629,83	—
jMobilePBC	397,76	477,64	557,12	567,95

V. CONCLUSÕES E TRABALHOS FUTUROS

Apresentamos a biblioteca criptográfica jMobilePBC, escrita em Java com compatibilidade JME, para desenvolvimento de aplicativos em dispositivos móveis, e compatibilidade J2SE para aplicações em servidor. Ela permite a implementação de protocolos criptográficos sobre curvas elípticas ordinárias e supersingulares, e sobre emparelhamentos simétricos e assimétricos. A jMobilePBC é distribuída como software livre e pretendemos que seja estendida com novos recursos.

É de interesse a inclusão de implementações de outros protocolos de criptografia de chave pública sobre curvas elípticas e sobre emparelhamentos bilineares. Também é de interesse que sejam acrescentadas codificações de técnicas otimizadas como as expostas mais recentemente em [25]. Tais otimizações foram codificadas em Java e se encontram disponíveis em [26].

AGRADECIMENTOS

Este trabalho foi financiado por Fapesp, sob os projetos de números 2008/06189-0 e 2008/50412-5.

REFERÊNCIAS

[1] JME. [Online]. Available: http://www.oracle.com/technetwork/java/javame.
[2] Bouncy castle. [Online]. Available: http://www.bouncycastle.org.
[3] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987. [Online]. Available: http://www.jstor.org/stable/2007884.
[4] V. S. Miller, "Use of elliptic curves in cryptography," in *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*. New York, NY, USA: Springer-Verlag New York, Inc.,1986, pp. 417–426.
[5] R. Sakai, K. Ohgishi, and M. Kasahara, "Cryptosystems based on pairing," in *Symposium on Cryptography and Information Security (SCIS2000)*. Okinawa, Japan: Inst. of Electronics, Information and Communication Engineers, 2000, pp. 26–28.
[6] R. Dutta, R. Barua, and P. Sarkar, "Pairing-based cryptographic protocols: A survey," Cryptology ePrint Archive, Report 2004/064, 2004.

[7] The pairing-based crypto lounge. [Online]. Available: http://www.larc.usp.br/pbarreto/pblounge.html.
[8] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 2001, pp. 213–229. [Online]. Available: http://eprint.iacr.org/2001/090.
[9] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, "Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions," *J. Cryptol.*, vol. 21, no. 3, pp. 350–391, 2008.
[10] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *ASIACRYPT 2003*, ser. Lecture Notes in Computer Science, vol. 2894. Springer, 2003, cryptology ePrint Archive, Report 2003/126, http://eprint.iacr.org/2003/126.
[11] FlexiProvider. [Online]. Available: http://www.flexiprovider.de
[12] Java crypto api. [Online]. Available: http://download.oracle.com/javase/1.5.0/docs/api/java/security/spec/EllipticCurve.html
[13] jPBC. [Online]. Available: http://sourceforge.net/projects/jpbc
[14] Secure sms. [Online]. Available: http:/secure-sms.googlecode.com
[15] E. Cruz, G. Pereira, R. R. Silva, and P. S. L. M. Barreto, "Construção de um sistema de sms seguro," in *Workshop de Trabalhos de Iniciação Científica e Graduação (WTICG'2008)*, Anais do Workshop de Trabalhos de Iniciação Científica e Graduação (WTICG'2008),. Gramado, RS:Anais do Workshop de Trabalhos de Iniciação Científica e Graduação (WTICG'2008), 2008 2008.
[16] Miyaji, Nakabayashi, and Takano, "New Explicit Conditions of Elliptic Curve Traces for FR-Reduction," *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 2001. [Online]. Available: http://citeseer.ist.psu.edu/miyaji01new.html
[17] jMobilePBC. [Online]. Available: svn://ccsl.ime.usp.br/lsd/trunk/jmobilepbc.
[18] Borboleta: A mobile telehealth system. [Online]. Available: http://ccsl.ime.usp.br/borboleta.
[19] J. Chung and M. A. Hasan, "Asymmetric squaring formulae," in *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 113–122. [Online]. Available: http://portal.acm.org/citation.cfm?id=1270377.1270455.
[20] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proceedings of SAC 2005, volume 3897 of LNCS*. Springer-Verlag, 2005, pp. 319–331.
[21] M. Wiener and R. Zuccherato, "Faster attacks on elliptic curve cryptosystems," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, S. Tavares and H. Meijer, Eds. Springer Berlin/Heidelberg, 1999, vol. 1556, pp. 631–631.
[22] A. Menezes, S. Vanstone, and T. Okamoto, "Reducing elliptic curve logarithms to logarithms in a finite field," in *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1991, pp. 80–89.
[23] C. M. Okida, "Protocolos de acordo de chaves baseados em emparelhamentos, para dispositivos móveis," Master's thesis, Universidade de São Paulo, 2011.
[24] B. Ustaoglu, "Obtaining a secure and efficient key agreement protocol from (h)mqv and naxos," *Des. Codes Cryptography*, vol. 46, no. 3, pp. 329–342, 2008.
[25] G. C. C. F. Pereira, M. A. Simplício, Jr., M. Naehrig, and P. S. L. M.Barreto, "A family of implementation-friendly bn elliptic curves," *J. Syst. Softw.*, vol. 84, pp. 1319–1326, August 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2011.03.083
[26] bnpairings. [Online]. Available: http://code.google.com/p/bnpairings

Cleber Okida é mestre em Ciência da Computação pelo Departamento de Ciência da Computação da Universidade de São Paulo (USP), São Paulo, em 2011. Obteve o título de Bacharel em Ciência da Computação pela Universidade Estadual Paulista "Julio de Mesquita Filho" (UNESP), Bauru, São Paulo, em 2007.

Denise Goya é doutora em Ciência em Ciência da Computação pelo Departamento de Ciência da Computação no Instituto de Matemática e Estatística da Universidade de São Paulo (USP), desde 2011. Obteve o título de Mestre e Bacharel em Ciência da Computação pela mesma instituição.

Routo Terada é Professor Titular no Departamento de Ciência da Computação no Instituto de Matemática e Estatística da Universidade de São Paulo (USP). Possui os seguintes títulos: PhD em Ciência da Computação pela Universidade de Wisconsin-Madison, Mestre em Matemática Aplicada pela Universidade de São Paulo, Engenheiro Eletricista-Eletrônico pela Escola Politécnica da Universidade de São Paulo.