Elaboração e Avaliação de um Modelo de Comunicação Direta Criptografada Entre Dispositivo Móvel e Servidor

Juliana G. de Souza[1], **Luciana A. F. Martimiano**[1]

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
CEP 87.020-900 – Maringá – PR – Brazil

juulianags@gmail.com, lafmartimiano@uem.br

Abstract. *The increasingly frequent use of mobile devices to remote access information highlights the need for ongoing studies on the security of data in transit. This paper seeks to establish and evaluate a model of communication between a server and a mobile device (client), considering the processing and memory limitations, that ensures data confidentiality through encryption. The model uses the cryptographic algorithms RSA and AES and evaluation thereof - in particular the performance in the generation of secret keys - and performed by an application developed using the Android platform.*

Resumo. *A utilização cada vez mais frequente de dispositivos móveis para acesso remoto de informações destaca a necessidade de contínuos estudos relativos à segurança dos dados em trânsito. O presente trabalho busca estabelecer e avaliar um modelo de comunicação entre um servidor e um dispositivo móvel (cliente), considerando as limitações de processamento e memória contidas neste, que garanta a confidencialidade dos dados mediante criptografia. O modelo utiliza os algoritmos criptográficos RSA e AES e a avaliação deste - em especial o desempenho na geração de chaves secretas - é realizada por meio de uma aplicação desenvolvida na plataforma Android.*

1. Introdução

Pesquisas afirmam que, dentre os domicílios brasileiros com acesso à Internet no ano de 2014, 80,4% o realizavam por meio de aparelho celular [IBGE 2016]. Esse acesso, tanto no Brasil como em todo o mundo, ocorre com aplicativos que realizam cada vez mais numerosas e diversificadas funções.

Muitos desses aplicativos armazenam e transmitem informações pessoais sigilosas. Cabe ao desenvolvedor proporcionar o máximo de confidencialidade, restringindo o acesso às informações apenas aos usuários devidos; integridade, impedindo a alteração indevida das informações; e disponibilidade, fornecendo os serviços propostos de maneira satisfatória. Além disso, é responsabilidade do desenvolvedor conhecer e compreender tecnologias diversas, afim de que as interações entre o sistema desenvolvido e outros sistemas, e até mesmo as interações internas, sejam realizadas de maneira eficiente e segura.

Em busca de vantagem competitiva e/ou proteção às suas operações, a maioria das organizações que realizam tráfego de informações sigilosas por meio de dispositivos móveis não divulga suas diretivas de segurança. Todavia, acompanhando o avanço da tecnologia e o aumento do poder de processamento e armazenamento dos computadores,

as estratégias de ataque se aperfeiçoam e se diversificam constantemente, exigindo que o tema continue sendo estudado e avaliado.

No decorrer deste trabalho, a principal característica de segurança abordada é a confidencialidade, garantida por meio de criptografia. O objetivo geral consiste em estabelecer e avaliar um modelo de comunicação entre um servidor e um dispositivo móvel, considerando as limitações de processamento e memória contidas neste. O modelo utiliza os algoritmos criptográficos RSA e AES e a avaliação deste, em especial o desempenho na geração de chaves secretas, é realizada por meio de uma aplicação *Android*.

O artigo está organizado da seguinte forma: a Seção 2 apresenta uma visão geral dos aspectos envolvidos na segurança, incluindo considerações sobre segurança no sistema *Android*; na Seção 3 são descritos o modelo de comunicação desenvolvido e a aplicação utilizada como estudo de caso; os resultados e a análise referentes à geração das chaves de criptografia são apresentados na Seção 4; e a Seção 5 apresenta as conclusões e as sugestões de trabalhos futuros.

2. Visão Geral da Segurança

O Ministério da Justiça define segurança como a "proteção dos ativos de informação contra perda, corrupção, destruição, acesso, uso e alteração indevidos ou não autorizados", e Segurança da Informação e Comunicações (SIC) como "ações que objetivam viabilizar e assegurar a disponibilidade, a integridade, a confidencialidade e a autenticidade das informações" [Brasil 2013]. Porém, chegar a um estado perfeito de segurança é impraticável [Six 2012], mas é possível identificar os riscos, amenizá-los e tornar viável a comunicação segura.

As ameaças podem estar relacionadas a diversos aspectos. *Softwares* maliciosos podem desencadear eventos não esperados. Existem ainda as falhas humanas, em que acidentes ou enganos das pessoas que interagem com o sistema desencadeiam consequências negativas. Além disso, existe a figura do agressor deliberado, que se torna uma ameaça tanto na tentativa de invasão e espionagem, quanto por objetivos de sabotagem e vandalismo [Marciano 2006].

Na busca pelo nível ideal de segurança é preciso pensar no maior número possível de características do *software* em questão. É preciso saber o que exatamente se quer proteger; as ameaças mais prováveis; a importância de cada parte do sistema; o grau desejável de segurança que esses dados exigem; os recursos disponíveis adequados a cumprir essa meta em se tratando de tempo, pessoas e recursos financeiros; e a expectativa dos indivíduos envolvidos com o projeto [Coelho et al. 2014]. Dessa maneira, é aconselhável que as preocupações com a segurança sejam de responsabilidade de todos os envolvidos, acrescentando, se possível, a realização de consultoria com especialistas [ABNT 2005].

Conforme a norma da ABNT [ABNT 2005], "convém que a política de computação móvel inclua os requisitos de proteção física, controles de acesso, técnicas criptográficas, cópias de segurança e proteção contra vírus". É de suma importância, juntamente, a análise da conformidade das operações realizadas no aplicativo com a legislação vigente.

Criptografia é uma técnica utilizada para a troca de mensagens fim a fim sem influência ou visualização de terceiros. Os algoritmos de criptografia são divididos en-

tre algoritmos de criptografia simétrica, em que o emissor e o receptor conhecem a chave utilizada para codificação e, por meio dela, realizam a decodificação da mensagem [Schneier 2000], como o AES (*Advanced Encryption Standard*); algoritmos de criptografia assimétrica, em que um par de chaves é gerado, uma privada e uma pública, para codificar e decodificar uma mensagem [Diffie and Hellman 1976], como o RSA (*Rivest - Shamir - Adleman*); e algoritmos de *hash*, que "mapeiam uma mensagem de tamanho variável em um valor de *hash* de tamanho fixo, ou um resumo da mensagem", sem a possibilidade de restaurar a mensagem inicial, como o SHA (*Secure Hash Algorithm*) [Stallings 2008].

2.1. Segurança na Plataforma *Android*

Desde o início do projeto *Android* considerou-se a segurança um ponto de suma importância. O planejamento ocorreu após a observação do funcionamento de outras plataformas, se aproveitando de padrões já existentes e procurando suprir necessidades detectadas. A arquitetura multicamadas é a principal estratégia de segurança da plataforma. Com uma estrutura semelhante a uma pilha, em cada nível é assumido que os anteriores estão seguros. No nível mais baixo são oferecidas as funções de segurança do *kernel* do *Linux* e definidas as permissões de comunicação entre os processos. Caso uma aplicação maliciosa queira prejudicar o sistema, a falta de interação com outros processos a impedirá de causar grandes estragos [ANDROID 2014].

O modelo de permissões é baseado no usuário. A cada aplicativo sendo executado é associado um número de identificação, juntamente com as informações do usuário, gerando um processo único, diferentemente do caso em que todos as aplicações são executadas com as mesmas permissões de usuário. Essa área da execução do processo é chamada de *sandbox* da aplicação e restringe as ações em seu próprio contexto, já que limita o contato com o sistema operacional e impede que outras aplicações e regiões de memória sejam afetadas.

3. Desenvolvimento da Aplicação

Com o objetivo de avaliar a geração de chaves dos algoritmos criptográficos e testar um modelo de comunicação seguro, foi realizado um estudo de caso em uma empresa real, que desejava o desenvolvimento de um aplicativo com as funcionalidades básicas de um sistema já existente. A funcionalidade principal consiste em receber informações pessoais do usuário e enviá-las a um servidor, que realiza uma análise de crédito baseada em inteligência artificial.

O projeto é composto por três módulos: aplicação cliente, comunicação e aplicação servidora. Os dados, e a tramitação desses, devem ser analisados de maneira apropriada em cada um dos módulos. Primeiramente, é preciso tratar a comunicação direta com o usuário, a recepção das informações e as primeiras ações necessárias a fim de preparar os dados, evitando informações incompletas, inconsistentes e dados faltantes. Essa análise é crucial para impedir que um agente malicioso realize injeção de código. Em um segundo momento, esses dados devem ser protegidos e transmitidos ao servidor sem intervenção de terceiros. Na aplicação servidora, por fim, as informações são processadas de acordo com os serviços solicitados e os resultados são enviados ao cliente.

Quanto ao ambiente de desenvolvimento, além da adição de senhas nas máquinas

envolvidas no projeto, estas devem ser mantidas em salas com restrição de acesso. Na empresa em que o estudo de caso foi realizado não é permitido que visitantes tenham acesso ao servidor local, localizado em sala separada e resfriada. As versões do projeto são mantidas no sistema *Apache Subversion*, um sistema para controle de versão de código livre. Desse modo, o projeto se mantém consistente e com possibilidade de recuperação. Os servidores são armazenados em dois lugares fisicamente distantes e possuem dados replicados, assegurando a existência de cópias de segurança. Todas as máquinas possuem sistemas de proteção a programas maliciosos e acessos indevidos.

3.1. Desenvolvimento da Comunicação

A comunicação entre a aplicação cliente (no dispositivo móvel) e a aplicação servidora (no servidor) é um ponto crítico em relação à segurança das informações. Os dados trocados partem dos extremos da comunicação e estão sujeitos a ameaças. À vista disso, com o intuito de garantir confidencialidade e integridade, técnicas criptográficas são utilizadas.

Na literatura, é possível identificar vários tipos de criptografia e algoritmos. Porém, definir como a criptografia será realizada não consiste apenas em implementar um algoritmo, é preciso selecioná-lo de acordo com alguns critérios e definir os passos do processo. Deve-se levar em consideração que as operações de codificação e decodificação serão realizadas em dispositivos móveis, que não possuem o mesmo poder de processamento e memória que os computadores em que esses algoritmos geralmente são implementados. Outra análise necessária é com relação aos avanços na tentativa de "quebrar" o algoritmo, ou seja, de obter a mensagem inicial ou texto claro sem possuir a chave, ou de deduzir a mesma.

Como o enfoque do modelo de comunicação situa-se na comunicação direta entre dispositivo móvel e servidor, a empresa decidiu não utilizar certificação digital e não envolver outras organizações. Essa condição imposta pela empresa em que o estudo de caso foi realizado, trouxe ao modelo uma possível vulnerabilidade a ataques de Man-in-The-Middle e semelhantes. Desse modo, não seria possível manter uma organização intermediária no sentido de auxiliar no estabelecimento da comunicação e troca de chaves. Caso a chave seja sempre a mesma e esteja armazenada tanto no servidor como no dispositivo, a troca de mensagens se torna mais simples. Existe, porém, a vulnerabilidade de manter a mesma chave em todos os dispositivos que utilizam o aplicativo, excluindo a possibilidade de alteração sem atualização total do aplicativo. Por outro lado, para operar uma chave variável é preciso tornar conhecida essa informação antes do início da comunicação, o que pode inutilizar todo o processo de codificação.

Denomine-se a chave da criptografia simétrica como chave secreta e o par de chaves da criptografia assimétrica como chave pública e chave privada. A solução proposta para a comunicação codificada consiste nas seguintes etapas, mostradas nas Figuras 2 (a), 2 (b) e 1:

1. No momento em que o usuário acessa o aplicativo, por meio da aplicação cliente, e realiza alguma operação que necessite acesso ao servidor, a mensagem A, a ser enviada, é criada.
2. Uma chave secreta é gerada e a mensagem A é codificada com ela, gerando uma mensagem codificada B.

3. A chave secreta é codificada com a chave pública não certificada do servidor, gerando uma mensagem C.
4. É enviada à aplicação servidora uma mensagem D composta de duas partes: a mensagem B e a mensagem C.
5. A aplicação servidora recebe a mensagem D e utiliza sua chave privada para decodificar a mensagem C e obter a chave secreta.
6. Por meio da chave secreta, a aplicação servidora decodifica a mensagem B, obtendo a mensagem A inicial.
7. A aplicação servidora processa a mensagem A e gera uma mensagem E de resposta.
8. A aplicação servidora utiliza a chave secreta para codificar a mensagem E, gerando uma mensagem F codificada, e a envia à aplicação cliente.
9. A aplicação cliente recebe a mensagem F e a decodifica com a chave secreta criada no início no processo.
10. A aplicação cliente interpreta a mensagem e exibe os resultados para o usuário.

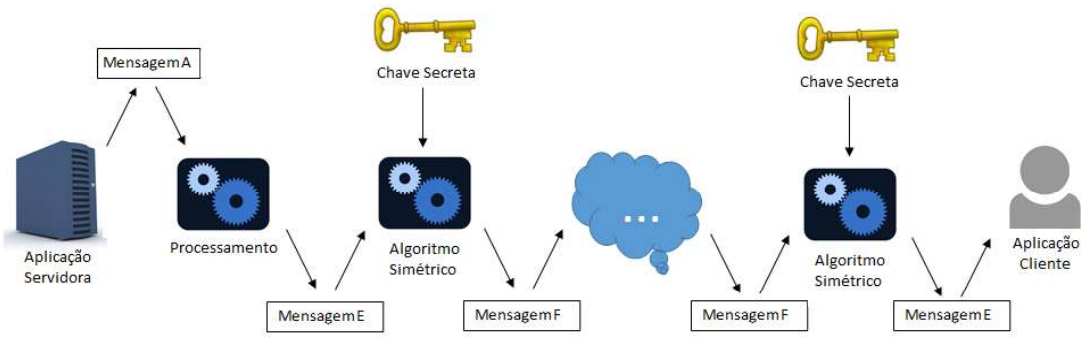

Figura 1. Procedimento de resposta da aplicação servidora à aplicação cliente.

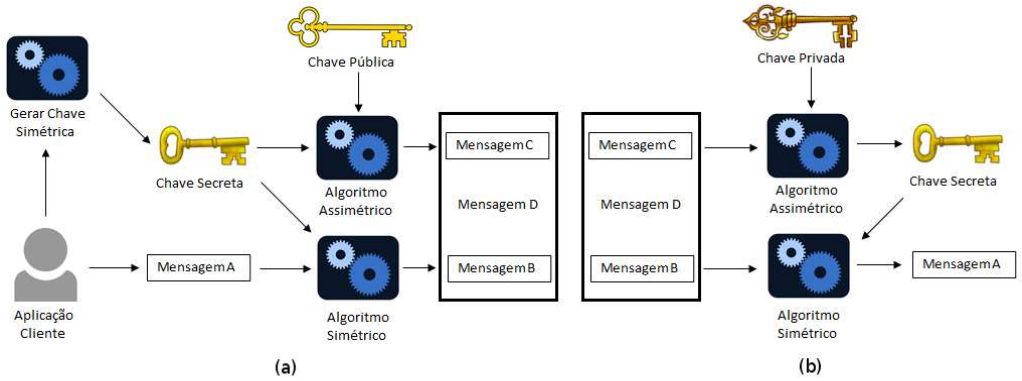

Figura 2. (a) Procedimento de envio de mensagem da aplicação cliente para a aplicação servidora. (b) Procedimento de recebimento da mensagem enviada pela aplicação cliente para a aplicação servidora.

Definido o modelo de criptografia, foram utilizados dois algoritmos considerando os seguintes requisitos:

- Não existirem registros ou previsões de "quebra" do algoritmo em um tempo razoável (todos os algoritmos podem ser quebrados por meio da geração de todas as chaves possíveis, porém, no caso dos algoritmos considerados seguros, isso levaria milhares de anos).
- Possibilidade de implementação nas linguagens Java e PHP, preferencialmente por meio de bibliotecas já existentes.
- Utilização de memória e processamento em quantidade razoável e com tempos de resposta aceitáveis em um dispositivo móvel.

Por meio das classes do pacote *Security*, implementou-se o algoritmo de chave assimétrica RSA, por ser um dos mais comuns atualmente e, portanto, mais testado, oferecendo maior confiabilidade. A desvantagem desse algoritmo, como é comum entre os algoritmos assimétricos, é o alto nível de processamento exigido. Isso ocorre devido à exigência de uma chave de no mínimo 2018 bits [Barker et al. 2016]. Porém, como o texto a ser codificado é pequeno, isso não inviabilizou o processo. Escolheu-se para a codificação simétrica o algoritmo AES, cujo uso é aconselhado pelo NIST (*National Institute of Standards and Technology*) e sua utilização constante atesta alto nível de segurança.

As funcionalidades da aplicação servidora incluem a codificação e decodificação realizadas pelo algoritmo AES-256 e decodificação assimétrica realizada pelo algoritmo RSA. No primeiro caso, a chave utilizada é recebida juntamente com mensagem da aplicação cliente. No caso do algoritmo RSA, é preciso analisar as questões concernentes à manutenção da chave privada, pois esta precisa ser mantida segura por mais tempo.

4. Análise dos Resultados

Por meio das definições de segurança, observou-se que não basta focar na ideia de assegurar o tráfego de dados na rede, mas, para um resultado positivo, se faz necessário analisar todos os componentes envolvidos no desenvolvimento da aplicação. O conceito de segurança abrange tanto as pessoas quanto os ambientes, as práticas seguras dentro de uma empresa, ameaças dentro do próprio dispositivo, acidentes ou intervenções intencionais, uso incorreto da criptografia, e diversos outros fatores e elementos que, de alguma forma, estão envolvidos com o sistema.

Tratando-se do *Android*, várias medidas de segurança foram embutidas na arquitetura da plataforma. Para um aplicativo malicioso ser capaz de acessar outros aplicativos dentro do dispositivo, sem as devidas permissões do usuário, por exemplo, seria preciso "driblar" as configurações do *kernel* do *Linux*. Por conseguinte, se o usuário se policiar e ler atentamente as solicitações de permissão dos aplicativos instalados em seu dispositivo, a probabilidade de acesso indevido às informações são mínimas. Quando a informação sai do dispositivo, porém, está sujeita a todo tipo de ameaça e condições adversas. O desenvolvedor se torna o responsável por estabelecer as medidas de segurança cabíveis, de acordo com o nível exigido pela aplicação. Não convém utilizar-se de todas as técnicas possíveis, gastar muitos recursos financeiros e computacionais, com a finalidade de garantir o máximo de segurança a informações que não necessitam de todas essas precauções. Em contrapartida, o descuido com algum dado sigiloso pode trazer consequências ruins a todos os envolvidos.

No tocante à escolha dos algoritmos adequados à criptografia, um dos fatores é a compatibilidade da linguagem empregada na implementação do aplicativo e a utilizada no

servidor em relação aos padrões de criptografia implementados. É preciso realizar testes com algoritmos diferentes em cada ponta da comunicação para definir os algoritmos que se comportam da mesma maneira em ambas, já que, dependendo das configurações dos *hardwares* e *softwares* utilizados no projeto, o desempenho pode sofrer alterações.

4.1. Geração da Chave Secreta

Elencaram-se duas abordagens para geração da chave secreta utilizada pelo algoritmo AES-256 na aplicação cliente: geração automática por meio da classe *SecretKey*, pertencente ao pacote *Javax.Crypto*, e geração manual por meio da geração de *bytes* aleatórios e um algoritmo de *hash*. Durante a geração da chave manual, é realizado um processamento em quatro etapas:

1. Um número aleatório N pertencente à uma determinada faixa é gerado.
2. É gerado um vetor V de *bytes* com N *bytes* aleatórios.
3. O vetor V é transformado em uma *string* S (cadeia de caracteres).
4. Um algoritmo de *hash* codifica S de modo a gerar uma chave secreta de tamanho padronizado de 256 *bits*.

Em uma chave composta por 256 *bits*, têm-se 7,2 * 10^{16} combinações possíveis. Esse número de possíveis chaves necessitaria, para ser gerado por meio de força bruta, um total de 2300 anos de processamento em um computador que realize um milhão de tentativas por segundo. Portanto, esse tamanho de chave pode ser considerado seguro para a aplicação.

Para identificar o número de *bytes* iniciais adequado, com baixo custo de implementação, para cada uma das etapas foram realizados testes de execução em dois dispositivos, doravante denominados dispositivos Z e M, respectivamente:

1. ZTE GV821 com processador *MediaTek* MT6516 / ARM926EJ-S rev 5 (*Single-Core*), memória interna de 256MB (100MB acessível pelo usuário) e tela de 2,4".
2. Moto G 2015 XT1544 Dual DTV com processador *Qualcomm*® *SnapdragonTM* 400 com 1,2 GHz *Quad-Core CPU*, memória interna de 16GB e tela de 5".

A medição de tempo foi realizada por meio do método *System.nanoTime()*, dessa forma, é possível registrar dois instantes de tempo dentro da aplicação e contabilizar a diferença entre eles. Em cada caso de teste, contabilizou-se o tempo em microssegundos da operação 12 vezes, eliminaram-se o maior e o menor valor e o resultado consistiu na média simples dos 10 valores restantes. A Tabela 1 contém os resultados da execução de três casos de teste para cada etapa e subetapa nos dispositivos Z e M, de acordo com o número de bytes definido no início do processo.

Para avaliação do comportamento da geração manual de chave secreta, os testes foram realizados com os respectivos valores de N: 2^4, 2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, em ambos os dispositivos. Porém, foi considerado suficiente para avaliar esse comportamento, os testes realizados com os valores de N contidos na Tabela 1.

É possível observar que a operação de geração de um número aleatório, que compreende a etapa 1, permanece com valores relativamente constantes, portanto, não influenciará na escolha do tamanho adequado da *string* aleatória a ser gerada. Na segunda etapa são necessárias duas operações: são gerados um vetor de *bytes* de tamanho N (etapa 2.1) e os N *bytes* aleatórios que o populam (etapa 2.2).

Tabela 1. Tempo em microssegundos das operações relacionadas à geração manual da chave secreta e codificação da mesma por meio do algoritmo RSA

	Dispositivo M					Dispositivo Z				
N bytes	2^6	2^{10}	2^{14}	2^{16}	2^{18}	2^6	2^{10}	2^{14}	2^{16}	2^{18}
Gerar um número aleatório entre 0 e o valor indicado										
Tempo	0.83	0.84	0.84	0.84	0.84	7277	7430	8715	7507	7661
em μs	0.84	0.85	0.85	0.85	0.84	8207	7638	9192	7938	8338
Etapa 1	0.84	0.85	0.87	0.85	0.85	8661	8784	9230	9038	9038
Gerar um vetor *bytes*, de acordo com o número de *bytes*										
Tempo	1.23	5.84	79.79	75.14	114.31	11.75	18.18	65.74	10192	95890.28
em μs	1.27	7.37	86.82	90.84	158.14	12.15	19.36	67.12	11309.82	95966.67
Etapa 2.1	1.5	8.52	132.17	111.98	811.47	12.27	28.47	68.91	13507.58	97948.43
Definir o vetor de *bytes* aleatórios de acordo com número de bytes										
Tempo	3.52	44.21	710.65	3004.86	11455.72	82.85	1260.64	27996.84	91493.45	373906.72
em μs	3.54	45.08	868.26	3581.59	12477.51	82.98	2375.29	30816.97	92373.43	376966.35
Etapa 2.2	3.9	45.88	931.14	4198.14	15635.38	83.51	2586.58	32369.28	100232.24	378060.39
Gerar uma *string* aleatória utilizando o comando 'for'										
Tempo	655.14	451441.85	-	-	-	40358.39	10741667.65	-	-	-
em μs	960.05	419311.64	-	-	-	44238.66	11393083.78	-	-	-
Etapa 3.1	1113.98	380543.89	-	-	-	45549.44	11401467.43	-	-	-
Gerar uma *string* aleatória utilizando o comando *new String(bytes)*										
Tempo	8.19	86.22	1278.77	4272.84	17801.98	2131.2	3434.25	31710.46	144877.58	617196.65
em μs	10.3	88.39	1474.02	4312.33	22366.36	2281.96	3732.75	32193.75	146171.34	619690.14
Etapa 3.2	10.37	93.69	1699.11	5172.67	23566.48	2310.92	4041.75	33157.32	154199.73	619850.81
Codificar com o algoritmo SHA-256, de acordo com o tamanho do vetor gerador da *string* aleatória										
Tempo	189.82	308.26	2727.65	7626.91	32833.57	6547.74	8262.78	17875.96	73675.34	298858.29
em μs	249.28	334.95	2850.53	8454.3	33647.35	6622.55	10398.92	22989.62	75996.06	300123.62
Etapa 4	326.33	366.41	2999.22	8646.07	38079.17	8068.58	10981.87	23160.88	77819.77	302769.41
Gerar a chave e codificar com o algoritmo RSA, utilizando *strings*										
Tempo em μs	4549.83	4634.15	7802.76	20117.01	69649.73	23304.95	18339.34	88381.48	287582.66	1210236.5
	4893.42	4715.01	8382.43	22608.76	70304.01	32927.24	19246.14	88651.31	288838.65	1226622.01
	6294.24	6801.11	9170.47	24188.35	71579.13	42622.91	28435.67	94475.22	298755.13	1228257.14
Gerar a chave e codificar com o algoritmo RSA, utilizando vetor de *bytes*										
Tempo em μs	2926.76	3145.29	4368.19	8565.3	22842.56	16597.35	14444.13	40750.84	111597.06	434900.58
	3501.89	4207.16	4823.47	9611.95	25318.18	17728.02	16420.09	42241.4	117569.09	437382.59
	4210.57	5208.31	5614.41	10589.31	29438.92	19477.22	26698.86	44506.43	122697.44	438112.93

Durante a terceira etapa, tendo o vetor de N *bytes*, é preciso montar a *string* aleatória a ser enviada à função *hash*. O primeiro método testado implementou um laço de repetição "for" que, a cada iteração, acrescenta um *byte* à *string* (etapa 3.1). Esse método foi mais custoso, levando mais de 10 segundos para o processamento de 1024 *bytes* no dispositivo Z. O segundo método testado foi a criação de uma *string* aleatória através do comando *new String(bytes)* (etapa 3.2), sendo *bytes*, o vetor de N *bytes* gerado anteriormente. Esse método apresentou melhores resultados que o anterior.

Tendo a *string* aleatória é realizada a quarta etapa e o resultado da função de *hash* é codificado por meio do algoritmo RSA. Por fim, também na Tabela 1, tem-se o tempo em microssegundos da operação completa de geração da chave e criptografia RSA. Calculou-se cada tempo com a utilização de um vetor de *bytes* de tamanho fixo, mas a operação de geração de um número aleatório N contabilizou-se variando entre 0 e 2^{12}. Isso porque, como já mostrado, o valor da operação se mantém semelhante até o limite de *bytes*.

Observou-se que as operações relativas à criação da *string* aleatória são tão custosas quanto ou mais custosas que as operações de criptografia. À vista disso, as operações de criptografia foram modificadas para trabalhar apenas com vetores de *bytes*, possibilitando a remoção da terceira etapa. Assim, o vetor de N *bytes* gerado é enviado à criptografia *hash* diretamente, bem como a chave gerada em *bytes* é enviada diretamente à criptografia RSA. Foram obtidos os resultados exibidos na Tabela 1.

Tabela 2. Informações estatísticas dos dados relativos ao tempo de execução (em μs) das operações de geração automática de chave e geração manual de chave nos dispositivos M e Z.

	Dispositivo M			Dispositivo Z		
Geração de Chave	**Média**	**Mediana**	**Desvio Padrão**	**Média**	**Mediana**	**Desvio Padrão**
Manual	16719,23	17121,79	1131,3	385397,9	386249,1	12709,8
Automática	53,37	52,7	7,4	488,1	486,773	13,3

Dessa forma, é possível realizar todas as operações em menos de meio segundo no dispositivo com menor poder de processamento e com o maior valor de *bytes* mostrado. Definiu-se, então, que a faixa de valores possíveis para N compreenderia de 2^{16} a 2^{18}, o que resulta em 196608 possibilidades de tamanhos para o vetor de *bytes*. É possível observar que, no dispositivo M, utilizando apenas *bytes*, obtém-se um ganho de tempo de aproximadamente 57%, no caso da geração de um vetor de *bytes* do tamanho mínimo possível e de aproximadamente 63,3% no caso de um vetor de tamanho máximo. No dispositivo Z o ganho é de aproximadamente 59,8% e 64,3%, respectivamente.

Seguindo a segunda abordagem, a geração automática da chave é feita por meio de quatro comandos:

```
1 KeyGenerator kg = KeyGenerator.getInstance("AES");
2 kg.init(256);
3 SecretKey sk = kg.generateKey();
4 byte[] chave = sk.getEncoded();
```

Foram realizadas 10 execuções da operação de geração automática de chave e 10 execuções da operação de geração manual de chave em cada dispositivo. A Tabela 2 contém informações estatísticas referentes a esses testes. É possível observar que, em ambos os dispositivos e abordagens, média e mediana são valores muito próximos e o desvio padrão é relativamente pequeno. Assim, podemos concluir que os resultados possuem pouca variação e a média se torna uma medida expressiva para o conjunto de dados. De acordo com a Tabela 2, a geração automática de chave tem um ganho médio em relação à geração manual de aproximadamente 99,87% para o dispositivo Z e 99,68% para o dispositivo M. Em vista disso, após todas as análises relativas à geração de chave, decidiu-se por aplicar a geração automática.

Esse resultado retrata que, apesar de ser possível a geração manual em tempo moderado, a utilização de bibliotecas e métodos nativos da linguagem podem poupar tempo tanto na implementação do aplicativo, quanto na execução. Apesar disso, é importante ter conhecimento do comportamento de soluções alternativas e das possibilidades dentro das limitações dos recursos disponíveis. Apesar de outros testes relativos à técnicas criptográficas terem sido realizados, escolheu-se a análise de geração de chaves para ser descrita, por compor um dos conjuntos de teste que melhor descreve o impacto de cada operação. Nela é possível observar que uma mudança de abordagem ou tipo de variável pode causar efeitos com alta significância na execução do processo.

5. Conclusão e Trabalhos Futuros

Na era da informação, a todo momento, dados estão sendo gerados e consumidos. Informações armazenadas, enviadas e recebidas por meio da Internet são pontos funda-

mentais para a tomada de decisão em todas as partes do planeta. Esse processo está cada vez mais intenso e rápido. Nesse cenário, a análise da segurança dessas informações se torna imprescindível.

Para o uso seguro de uma informação é preciso que esta tenha partido do emissor e chegado ao receptor nas condições devidas, podendo ser armazenada por tempo indeterminado nesse caminho. Com a popularização dos dispositivos móveis, essa movimentação da informação passou a ser feita, em grande parte, por meio deles. Surgem então questionamentos sobre como a segurança da informação deve ser tratada nessas tecnologias e quais seriam os problemas envolvidos.

Deste ponto parte o presente trabalho. Em princípio, buscou-se conhecer o que é preciso saber, em relação à segurança, antes de avaliar ou implementar um aplicativo para dispositivo móvel. Tendo as noções gerais, a segurança da plataforma de desenvolvimento é avaliada. A plataforma utilizada na aplicação desenvolvida proporciona alto nível de segurança das informações processadas localmente no dispositivo, devido ao funcionamento de seu sistema operacional e esquema de permissões, o que rende à comunicação externa maior foco de preocupação do desenvolvedor.

Observou-se que as características próprias da linguagem influenciam diretamente no desempenho das operações. É preciso definir uma maneira adequada para o desenvolvimento que viabilize a execução em tempo aceitável. Na linguagem Java, utilizada pela plataforma *Android*, realizar operações com dados no formato *string* pode ser muito custoso, como constatado nos testes de geração de chaves. Utilizar bibliotecas nativas é uma boa estratégia para otimizar os processos, pois, na maior parte dos casos, possuem implementação eficiente e segura, economizando tempo de desenvolvimento e testes mais detalhados, conferindo ao sistema mais credibilidade e facilidade de manutenção.

O trabalho descreveu possibilidades para a implementação da comunicação direta entre um dispositivo móvel e um servidor, sem a interferência de outras organizações. A confidencialidade da informação é proporcionada pelo modelo de criptografia desenvolvido, que emprega criptografia simétrica, por meio do algoritmo AES, e criptografia assimétrica, por meio do algoritmo RSA.

Como possíveis trabalhos futuros podem-se apontar o maior detalhamento comparativo do desempenho dos algoritmos de criptografia em dispositivos móveis, além do acréscimo de medidas de segurança ao projeto, como certificação digital. Outras possibilidades envolvem a análise de segurança de aplicativos utilizados em outros sistemas operacionais, como IOS, desenvolvido pela empresa *Apple*, e *Windows Phone*, desenvolvido pela *Microsoft*.

Referências

ABNT (2005). Nbr iso/iec 27002: Tecnologia da informação - técnicas de segurança - código de prática para a gestão da segurança da informação. *ABNT*.

ANDROID, O. S. P. (2014). Android security overview.

Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M. (2016). *Computer Security: Recommendation for Key Management – Part 1: General (Revised)*. NIST, 4 edition.

Brasil, M. D. J. (2013). Política de segurança da informação e comunicações do ministério da justiça: Portaria nº. 3.530, de 3 de dezembro de 2013.

Coelho, F. E. S., de Araújo, L. G. S., and Bezerra, E. K. (2014). Gestão da segurança da informação: Nbr 27001 e nbr 27002. *Rnp/esr*.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654.

IBGE (2016). Pnad tic: em 2014, pela primeira vez, celulares superaram microcomputadores no acesso domiciliar à internet.

Marciano, J. L. P. (2006). Segurança da informação: Uma abordagem social.

Schneier, B. (2000). *Secrets & Lies: Digital Security in a Networked World*. Wiley Computer Publishing, United States of America.

Six, J. (2012). *Segurança de Aplicativos Android*. Novatec, São Paulo.

Stallings, W. (2008). *Criptografia e Segurança de Redes: Princípios e Práticas*. Pearson Prentice Hall, São Paulo, 4 edition.

