**Entendendo segurança no Android**

**Melhore a segurança dos aplicativos com ambientes de simulação, registro nos aplicativos e permissões**

Ao desenvolver aplicativos Android, é preciso lidar com uma série de aspectos relacionados à segurança, incluindo processos de aplicativos e ambientes de simulação, compartilhamento de código e dados, proteção do sistema por meio de assinatura do aplicativo e uso de permissões. Destaque esses aspectos de segurança no desenvolvimento de aplicativos Android ao trabalhar com o código de amostra.

0http://dw1.s81c.com/developerworks/i/v17/dw-cmts-arrow.png [Comentários](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#icomments)

[C. Enrique Ortiz](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#authorN1001C), Mobility Technologist and Writer, J2MEDeveloper.com

22/Dez/2010

* [expand](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#toggle)**Índice**



Desenvolva e implemente seu próximo aplicativo na plataforma de cloud do IBM Bluemix.

[**Comece seu  
trial gratuito**](https://developer.ibm.com/sso/bmregistration?lang=pt_BR&ca=dwbrazil-_-bluemix-_-x-androidsecurity-_-sidebar)

**Visão geral**

O Android consiste em uma estrutura de aplicativo, bibliotecas de aplicativo e um tempo de execução com base em máquina virtual Dalvik, todos eles em execução em um kernel do Linux® . Ao tirar proveito do kernel Linux, o Android obtém uma série de serviços do sistema operacional, incluindo o gerenciamento de processos e de memória, uma pilha de rede, drivers, uma camada de abstração de hardware e, relacionados ao tópico deste artigo, serviços de segurança.

**Pré-requisitos**

**Acrônimos usados frequentemente**

* ADT: Android Development Tools
* API: Interface de programação de aplicativos
* IDE: Ambiente de Desenvolvimento Integrado
* JDK: Java Development Kit
* URL: Identificador Uniforme de Recursos
* XML: Linguagem de Marcação Extensível

Para acompanhar este artigo, é preciso ter as seguintes qualificações e ferramentas:

* Conhecimento básico da tecnologia Java™ e como usar o Eclipse (ou seu IDE favorito)
* Java Development Kit (é requerida a versão 5 ou 6)
* Eclipse (versão 3.4 ou 3.5)
* Android SDK e plug-in ADT

Para fazer o download e obter informações de configuração, consulte a seção [Recursos](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#resources)no final deste artigo.

[**Voltar para parte superior**](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#ibm-pcon)

**Ambientes de simulação, processos e permissões**

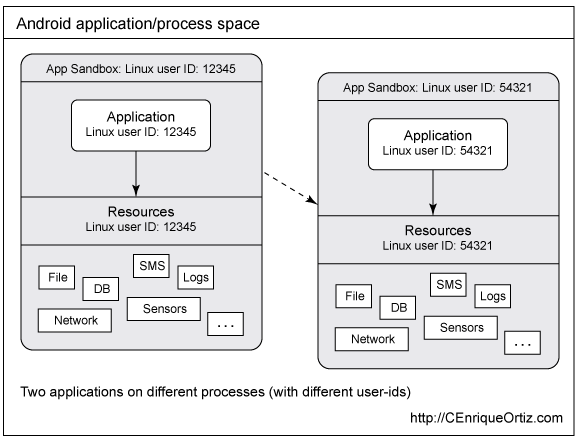
**ID do usuário: Linux versus Android**

Enquanto que um ID de usuário no Linux identifica um dado usuário, no Android, um ID de usuário identifica um aplicativo. IDs de usuário são atribuídos quando o aplicativo é instalado e permanecem pela vida útil do aplicativo no dispositivo. Permissões dizem respeito a permitir ou restringir o acesso do aplicativo (em vez de usuários) aos recursos do dispositivo.

O Android usa o conceito de um ambiente de simulação para executar a separação e as permissões entre aplicativos a fim de permitir ou negar o acesso de um aplicativo aos recursos do dispositivo, como arquivos e diretórios, a rede, os sensores e as APIs em geral. Para isto, o Android usa facilidades do Linux, como segurança em nível de processo, IDs de usuário e de grupo associados com o aplicativo e permissões para impor quais operações um aplicativo tem permissão de realizar.

Conceitualmente, um ambiente de simulação pode ser representado como na [Figura 1](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig1).

**Figura 1. Dois aplicativos Android, cada um em seu próprio ambiente de simulação ou processo básico**



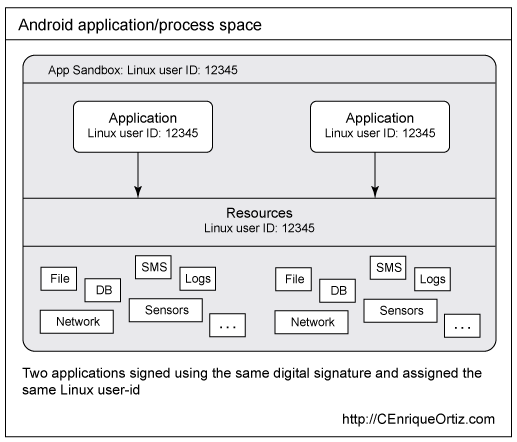
O aplicativo Android é executado em seu próprio processo do Linux e lhe é atribuído um ID de usuário exclusivo. Por padrão, os aplicativos são executados dentro de um processo básico de ambiente de simulação sem permissões atribuídas, evitando, desta forma, que tais aplicativos acessem o sistema ou os recursos. Aplicativos Android podem solicitar permissões, no entanto, por meio do arquivo de manifesto do aplicativo.

Aplicativos Android podem permitir o acesso a seus recursos por outros aplicativos ao:

* Declarar as permissões de manifesto apropriadas
* Executar no mesmo processo com outros aplicativos confiáveis e, portanto, compartilhar o acesso a seus dados e código

O último caso é ilustrado na [Figura 2](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig2).

**Figura 2. Dois aplicativos Android executando no mesmo processo**



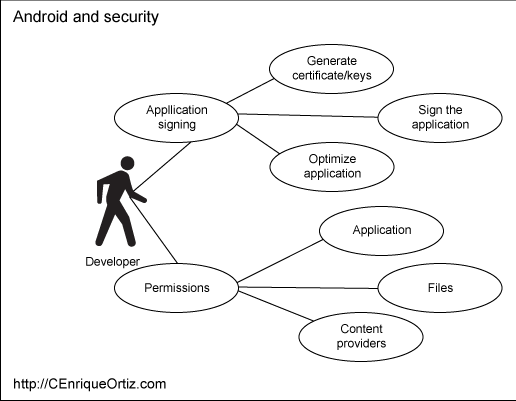
Diferentes aplicativos podem ser executados no mesmo processo. Para esta abordagem, é preciso primeiro assinar esses aplicativos usando a mesma chave privada e, a seguir, atribuir a eles o mesmo ID de usuário do Linux usando o arquivo de manifesto, definindo o atributo do manifestoandroid:sharedUserId com o mesmo valor/nome.

[**Voltar para parte superior**](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#ibm-pcon)

**Casos de uso de desenvolvedores**

[A Figura 3](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig3) ilustra uma série de *casos de uso* relacionados a segurança, encontrados ao desenvolver aplicativos Android.

**Figura 3. Áreas de segurança presentes ao programar aplicativos Android**



* A assinatura do aplicativo ou código é o processo de gerar chaves privadas e públicas e certificados de chave pública, assinar e otimizar o aplicativo.
* Permissões é um mecanismo de segurança da plataforma Android para permitir ou restringir o acesso do aplicativo a APIs e recursos restritos. Por padrão, aplicativos Android não têm permissões concedidas, tornando-os seguros ao não permitir que tenham acesso a APIs ou recursos protegidos no dispositivo. Permissões devem ser solicitadas, permissões customizadas devem ser definidas e provedores de conteúdo e arquivos protegidos. Assegure-se de verificar, impingir, conceder e revogar permissões em tempo de execução.

A seguir, veja com mais detalhes cada uma das áreas de segurança.

[**Voltar para parte superior**](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#ibm-pcon)

**Assinatura do aplicativo**

Todos os aplicativos Android devem ser assinados. A assinatura do aplicativo ou código é o processo de assinar digitalmente um dado aplicativo usando uma chave privada para:

* Identificar o autor do código
* Detectar se o aplicativo foi alterado
* Estabelecer a confiança entre aplicativos

Com base nesse relacionamento confiável, os aplicativos podem compartilhar código e dados de forma segura.

Aplicativos assinados usando a mesma assinatura digital podem conceder um ao outro permissões para acessar APIs com base em assinatura e, também, executar no mesmo processo se compartilharem IDs de usuário, permitindo o acesso ao código e dados um do outro.

A assinatura do aplicativo começa com a geração de um par de chaves privada e pública e um certificado relacionado de chave pública, também conhecido como certificado de chave pública.

Ao trabalhar com aplicativos Android, é possível construir aplicativos em modo *debug* e *release*:

* Aplicativos construídos usando as ferramentas de construção Android (linha de comando e Eclipse ADT) são automaticamente assinados usando uma chave privada de depuração. Esses aplicativos são conhecidos como aplicativos em debug-mode. Aplicativos em debug-mode são usados para teste e não devem ser distribuídos. Note que aplicativos não assinados ou assinados usando uma chave privada de depuração não podem ser distribuídos por meio do Android Market.
* Quando você estiver pronto para fazer o release de seu aplicativo, é preciso construir uma versão em modo release dele, o que significa assinar o aplicativo com sua chave privada.

A assinatura de código no Android é feita de uma maneira muito mais simples do que em outras plataformas de dispositivos remotos. No Android, o certificado pode ser autoassinado, ou seja, não há necessidade de uma autoridade de certificação. Esta abordagem simplifica o processo de publicação e os custos relacionados.

A seguir, abordarei como assinar manualmente aplicativos Android a partir da linha de comando e usando o ADT. Não discutirei neste artigo uma terceira abordagem, a qual usa Ant.

**Criando manualmente as chaves privada e pública e o certificado de chave pública**

Lembre-se de que aplicativos em modo de depuração são assinados automaticamente pelas ferramentas de construção usando a chave/certificado de depuração. Para assinar um aplicativo em release-mode, é preciso primeiro gerar o par de chaves privada e pública e o certificado de chave pública. É possível assinar seu aplicativo manualmente usando o ADT. Em ambas as abordagens, o utilitário de gerenciamento de chave e certificado keytool do Java Developer Kit (JDK) é usado.

Para gerar as informações de chaves privada e pública manualmente, use keytool a partir da linha de comando, como na [listagem 1](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#list1).

**Listagem 1. Usando keytool para gerar chaves privada/pública e certificado**

keytool -genkey -v -alias <alias\_name> -keystore

<keystore.name> -keyalg RSA -keysize 2048 -validity <number of days>

**NOTA:**[A listagem 1](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#list1) presume que o JDK esteja instalado em seu computador e que o caminho JAVA\_HOME esteja definido adequadamente para apontar para seu diretório JDK (consulte [Recursos](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#resources)para obter informações de download e configuração).

Na [listagem 1,](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#list1), o -genkey indica uma entrada de par de chaves pública e privada, bem como uma cadeia de certificados X.509 v1 autoassinadas de um único elemento que inclui a chave pública gerada. -v indica o modo detalhado. -alias é o alias que irá ser usado para a entrada de keystore que armazena a chave privada e o certificado gerados. -keystore indica o nome do keystore que será usado. -keyalg é o algoritmo que será usado para gerar o par de chaves. -keysize é o tamanho da chave que será gerado, em que o padrão é 1024, mas o recomendado é 2048. E -validity é o período de validade em dias. Um valor maior do que 1000 é recomendado.

**NOTA:**Depois de gerar suas chaves, é muito importante salvaguardar sua chave privada. Não compartilhe sua chave privada e não a especifique na linha de comando ou em scripts; note que keytool e jarsigner solicitam a senha. Para ver esta e outras dicas, consulte "Securing Your Private Key" no web site Android Developers (consulte [Recursos](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#resources)para obter um link).

Keytool solicita seu nome e sobrenome, empresa, cidade, estado, país para o qual um nome distinto X.500 é gerado (consulte [Recursos](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#resources)para obter mais informações) e senhas para proteger a chave privada e para proteger o keystore em si.

Para o período de validade, assegure-se de usar um período que exceda a vida útil esperada do aplicativo e aplicativos relacionados. Se estiver publicando seu aplicativo no Android Market, o período deverá terminar depois de 22 de outubro de 2033; caso contrário, não será possível fazer seu upload. Além disso, ter certificados de longa duração torna sua vida mais fácil ao atualizar seus aplicativos. Felizmente, o Android Market impinge certificados de longa duração para ajudá-lo a evitar tais situações.

**Assinando seu aplicativo manualmente**

A seguir, assine o aplicativo não assinado usando a ferramenta jarsigner , que é parte do JDK:

jarsigner -verbose -keystore <keystore.name> <my\_application.apk> <alias\_name>

No código precedente, -verbose indica o modo detalhado e -keystore indica o nome do keystore a usar. A seguir, vem o nome do aplicativo (.apk) e, por último, está o alias a usar para a chave privada.

O Jarsigner solicita a senha para usar o keystore e a chave privada.

Aplicativos podem ser assinados diversas vezes usando diferentes chaves e aplicativos assinados com a mesma chave privada podem estabelecer um relacionamento confiável entre si e executar no mesmo processo, bem como compartilhar código e dados.

**Otimizando seu aplicativo manualmente**

A última etapa no processo de assinatura é otimizar o aplicativo, para que os limites de dados sejam alinhados em memória com respeito ao início do arquivo, uma técnica que ajuda a melhorar o desempenho em tempo de execução e a utilização de memória. Para alinhar o aplicativo, usezipalign:

zipalign -v 4 your\_project\_name-unaligned.apk your\_project\_name.apk

No código precedente, -v indica o modo detalhado. O número 4 indica o uso de alinhamento de quatro bytes (sempre use quatro bytes). O próximo argumento é o nome do arquivo do aplicativo assinado de entrada (.apk), que deve estar assinado com sua chave privada e o último argumento é o nome do arquivo de saída; se estiver substituindo um aplicativo existente, adicione -f.

**Verificando manualmente se o aplicativo foi assinado**

Para verificar se o aplicativo foi assinado, use o Jarsigner, desta vez passando o sinalizador -verify :

jarsigner -verify -verbose -certs my\_application.apk

No código precedente, -verify indica que é para verificar o aplicativo; -verbose indica o modo detalhado; -certs indica que deve ser mostrado o campo CN de quem criou a chave e o último argumento é o nome do pacote do aplicativo Android a verificar.

**NOTA:**Se o CN tiver o valor "Android Debug", significa que o aplicativo foi assinado usando a chave de depuração e, portanto, não pode ser publicado; lembre-se de usar uma chave privada se planejar publicar seu aplicativo no Android Market.

Você acabou de ver como criar as chaves privada e pública manualmente e como assinar e otimizar o aplicativo. A seguir, veja como usar o Eclipse ADT para criar automaticamente as chaves privada e pública e para assinar e otimizar o aplicativo para você.

[**Voltar para parte superior**](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#ibm-pcon)

**Usando o Eclipse ADT para criar chaves e certificado e para assinar e otimizar seu aplicativo**

Para gerar as chaves usando o Eclipse ADT, é preciso exportar o aplicativo. Existem dois métodos para exportar aplicativos do Eclipse:

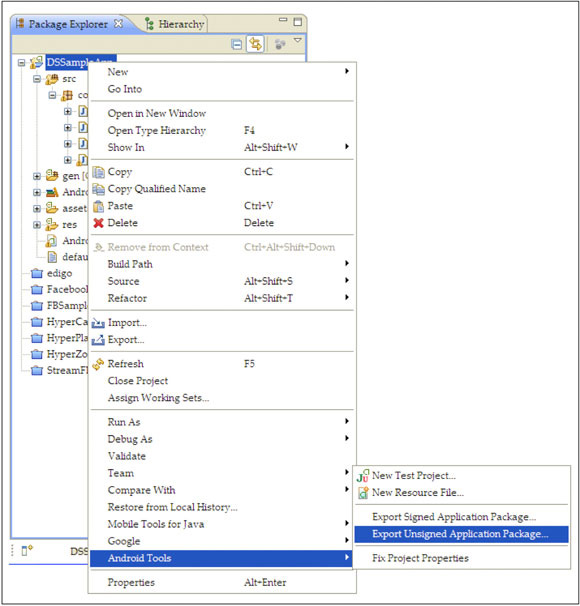
* Exportar uma versão *não assinada* do aplicativo que você possa assinar manualmente
* Exportar uma versão *assinada* do aplicativo, onde o ADT realiza todas as etapas para você

**Exportando um aplicativo não assinado**

É possível exportar uma versão não assinada de seu aplicativo que deverá ser assinada manualmente. Ou seja, é preciso executar manualmente o keytool (para gerar as chaves, como descrito anteriormente) e o Jarsigner (para assinar o aplicativo) e otimizar seu aplicativo usando a ferramenta zipalign, como explicado anteriormente.

Para exportar uma versão não assinada do aplicativo usando o ADT, clique com o botão direito do mouse no projeto e selecione **Android Tools>Export Unsigned Application Package** (consulte a [Figura 4](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig4)).

**Figura 4. Exportando o aplicativo não assinado**



Depois de selecionado, o ADT solicita o diretório para onde exportar o aplicativo não assinado. Lembre-se de que, uma vez que o aplicativo seja exportado, é preciso assinar e otimizar manualmente o aplicativo, como abordado anteriormente.

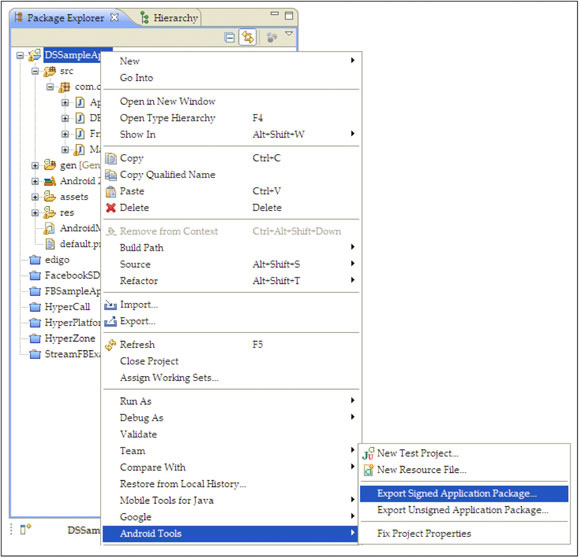
**Exportando um aplicativo assinado**

Com o Eclipse ADT, é possível exportar uma versão assinada do aplicativo. Usando esta abordagem, o ADT solicita o seguinte:

* As informações necessárias para usar um KeyStore existente ou criar um novo KeyStore protegido
* As informações necessárias para criar uma chave privada protegida
* As informações necessárias para gerar o certificado de chave pública

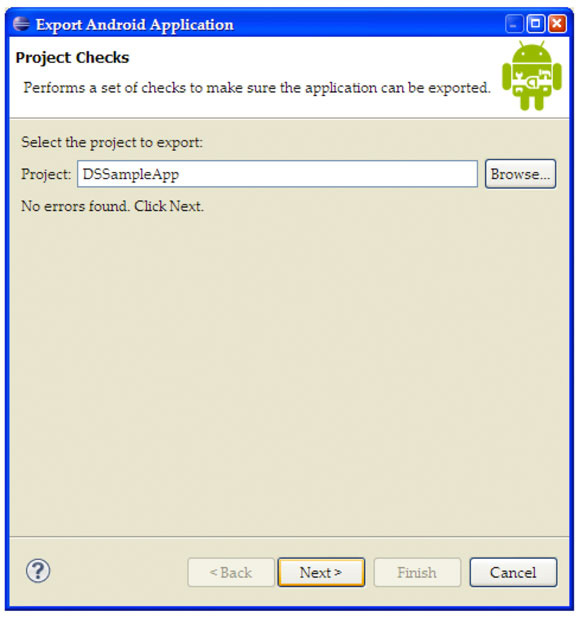
Para exportar um aplicativo assinado, clique com o botão direito do mouse no projeto, mas, desta vez, selecione o item de menu **Android Tools->Export Signed Application Package**, conforme ilustrado na [Figura 5](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig5).

**Figura 5. Exportando um aplicativo assinado**



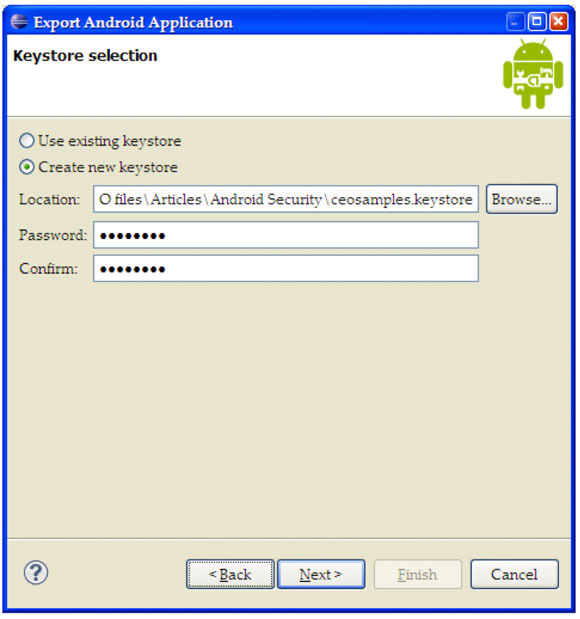
Neste ponto, o assistente de exportação é executado, como mostra a [Figura 6.](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig6).

**Figura 6. Assistente de exportação**



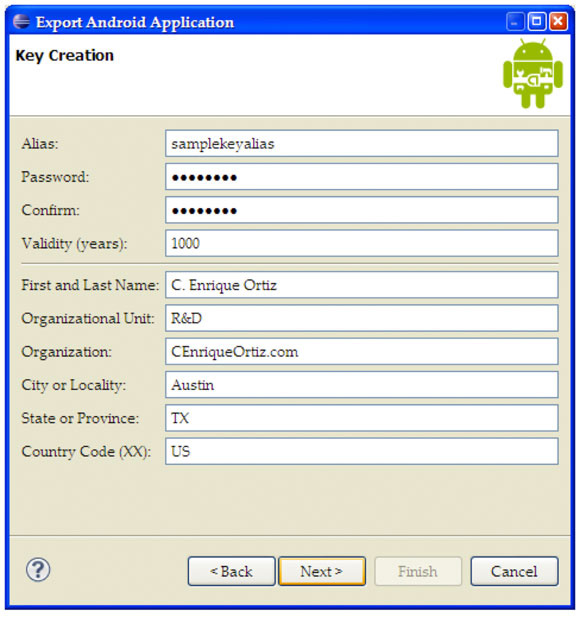
Na [Figura 7](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig7), selecione um keystore existente ou crie um novo e as credenciais a usar.

**Figura 7. Assistente de exportação: seleção do Keystore**



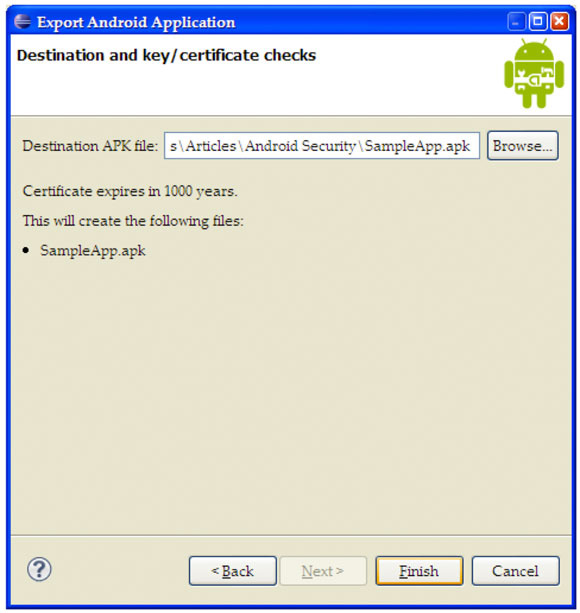
Na [Figura 8](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig8), insira as informações para criar a chave privada e o certificado digital.

**Figura 8. Assistente de exportação: Criando a chave privada e o certificado digital**



Na [Figura 9](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig9), insira o caminho e o nome do arquivo de destino e verifique o período de validade.

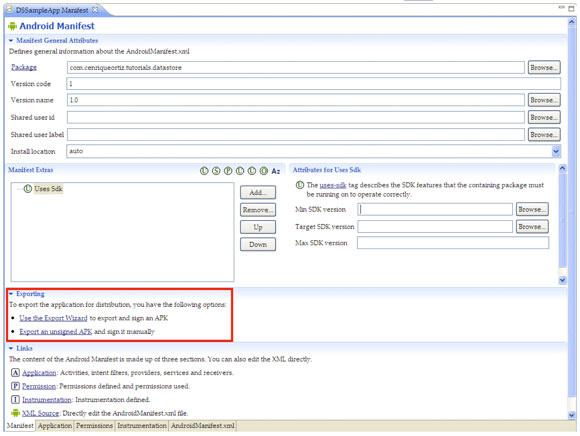
**Figura 9. Inserindo o caminho e o nome do arquivo de destino**



Quando concluir, você terá um aplicativo em modo release assinado e otimizado que poderá publicar.

Alternativamente, também é possível chamar o assistente de exportação usando a Android Manifest Tool, como mostra a [Figura 10](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#fig10).

**Figura 10. Chamando o assistente de exportação usando a ferramenta Android Manifest**



Depois que os aplicativos estiverem assinados, a próxima etapa é definir no manifesto as permissões necessárias pelo aplicativo. Este processo é descrito a seguir.

Note que o web site Android Developer tem documentação muito boa sobre assinatura de aplicativos que está sempre atualizada à medida que novos releases da plataforma Android tornam-se disponíveis (consulte [Recursos](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#resources)para obter mais informações).

[**Voltar para parte superior**](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#ibm-pcon)

**Usando permissões**

Permissões é um mecanismo de segurança da plataforma Android para permitir ou restringir o acesso a aplicativos para APIs e recursos restritos. Por padrão, aplicativos Android não têm permissões concedidas, tornando-os seguros ao não permitir que tenham acesso a APIs ou recursos protegidos no dispositivo. Permissões são solicitadas pelo aplicativo por meio do arquivo de manifesto e concedidas ou não pelo usuário durante a instalação.

O Android define uma longa lista de permissões de manifesto, protegendo vários aspectos do sistema ou outros aplicativos. Para solicitar permissão, declare um atributo <user-permission> no arquivo de manifesto:

<uses-permission android:name="string" />

onde android:name especifica o nome da permissão.

Para obter uma lista de todas as permissões de manifesto definidas pelo Android, consulte a página Manifest.permisson. [A Listagem 2](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#list2) é um exemplo de um arquivo de manifesto solicitando permissão para usar a Internet e permissão para gravar no armazenamento externo:

**Listagem 2. Declarando (solicitando) uma permissão**

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

android:versionCode="1"

android:versionName="1.0"

package="com.cenriqueortiz.tutorials.datastore"

android:installLocation="auto">

<application

:

:

:

</application>

<uses-permission

android:name="android.permission.INTERNET"/>

<uses-permission

android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>

</manifest>

Aplicativos podem definir suas próprias permissões customizadas para proteger recursos de aplicativo. Outros aplicativos que queiram acessar os recursos protegidos de um aplicativo deverão solicitar as permissões apropriadas por meio de seu próprio arquivo de manifesto. [A Listagem 3](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#list3)mostra um exemplo de como definir permissões.

**Listagem 3. Declarando uma permissão customizada**

<permission

xmlns:android="http://schemas.android.com/apk/res/android"

android:name="com.cenriqueortiz.android.ACCESS\_FRIENDS\_LIST"

android:description="@string/permission\_description"

android:label="@string/permission\_label"

android:protectionLevel="normal"

<

</permission<

Na [Listagem 3](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#list3), uma permissão de cliente é definida especificando os atributos mínimos name, description, labele protectionLevel. Outros atributos podem ser definidos, mas não são abordados aqui.

De interesse especial é o atributo android:protectionLevel , que indica o método que o sistema deverá seguir ao conceder (ou não) uma dada permissão para um aplicativo solicitante. Os níveis de proteção incluem *normal*, que concede automaticamente permissões (mas o usuário sempre poderá revisá-las antes da instalação), concedendo permissões com base na assinatura (ou seja, se o aplicativo solicitante estiver assinado como mesmo certificado) e *dangerous*, indicando que a permissão dá acesso a dados privados ou tem outro impacto negativo potencial. Para obter mais informações sobre o atributo de manifesto <permission> , consulte a página <permission> (consulte [Recursos](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#resources)).

Os aplicativos podem restringir o acesso ao aplicativo e aos componentes de sistema que usa, como Activity, Service, Content Provider e Broadcast Receiver. Esta restrição é feita facilmente definindo o atributo android:permission , como mostra a [Listagem 4](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#list4). Este nível de proteção possibilita que o aplicativo permita ou restrinja o acesso de outros aplicativos ao recurso do sistema.

**Listagem 4. Definindo uma permissão para uma atividade**

<activity

android:name=".FriendsListActivity"

android:label="Friends List">

android:permission="com.cenriqueortiz.android.ACCESS\_FRIENDS\_LIST"

<intent-filter>

:

:

</intent-filter>

</activity>

[**Voltar para parte superior**](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#ibm-pcon)

**Provedores de conteúdo e permissões de arquivos**

Provedores de conteúdo expõem um URI público que identifica de forma exclusiva seus dados (consulte [Recursos](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#resources)). Para proteger tais provedores de conteúdo, o responsável pela chamada, ao iniciar ou retornar um resultado de uma atividade, pode definir aIntent.FLAG\_GRANT\_READ\_URI\_PERMISSION e asIntent.FLAG\_GRANT\_WRITE\_URI\_PERMISSION para conceder a permissão da atividade recebida para acessar o URI específico dos dados no intento.

Arquivos de aplicativos são protegidos por padrão. Os arquivos são protegidos com base em seus IDs de usuário e, portanto, acessíveis somente pelo aplicativo proprietário, que tem o mesmo ID de usuário. Como abordado anteriormente, aplicativos que compartilham o mesmo ID de usuário (e são assinados usando o mesmo certificado digital) executam no mesmo processo e, portanto, compartilham o acesso a seus aplicativos.

Aplicativos podem permitir o acesso a seus arquivos para outros aplicativos ou processos. Esta permissão é feita indicando o modo de operaçãoMODE\_WORLD\_READABLE e MODE\_WORLD\_WRITEABLE apropriado para permitir o acesso de leitura ou gravação ao arquivo ou MODE\_PRIVATEpara abrir o arquivo em modo privado. É possível especificar um modo operacional com os seguintes métodos ao criar ou abrir arquivos:

* getSharedPreferences(filename, operatingMode)
* openFileOutput(filename, operatingMode)
* openOrCreateDatabase(filename, operatingMode, SQLiteDatabase.CursorFactory)

[**Voltar para parte superior**](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#ibm-pcon)

**APIs de permissão em tempo de execução**

O Android fornece APIs para verificar, impingir, conceder e revogar permissões em tempo de execução. Essas APIs são parte da classeandroid.content.Context , que fornece informações globais sobre um ambiente de aplicativos. Por exemplo, se quiser tratar permissões de forma tolerante, é possível determinar se seu aplicativo recebeu acesso à Internet (consulte a [Listagem 5](http://www.ibm.com/developerworks/br/library/x-androidsecurity/#list5)).

**Listagem 5. Usando uma API de permissão em tempo de execução para verificar permissões**

if (context.checkCallingOrSelfPermission(Manifest.permission.INTERNET)

!= PackageManager.PERMISSION\_GRANTED) {

// The Application requires permission to access the

// Internet");

} else {

// OK to access the Internet

}