TinySec: User Manual

Chris Karlof Naveen Sastry David Wagner
{ckarlof, nks, daw}@cs.berkeley.edu

June 17, 2004

1 Introduction

We introduce TinySec, a link layer encryption mechanism which is meant to be the first part in a suite of security solutions for tiny devices. The core of TinySec is an efficient block cipher and keying mechanism that is tightly coupled with the Berkeley TinyOS radio stack. TinySec currently utilizes a single, symmetric key that is shared among a collection of sensor network nodes. Before transmitting a packet, each node first encrypts the data and applies a Message Authentication Code (MAC), a cryptographically strong unforgeable hash to protect data integrity. The receiver verifies that the packet was not modified in transit using the MAC and then deciphers the message.

There are four main aims of TinySec:

- Access control. Only authorized nodes should be able to participate in the network. Authorized nodes are designated as those nodes that possess the shared group key.
- Integrity. A message should only be accepted if it was not altered in transit. This prevents, for example, man-in-the-middle attacks where an adversary overhears, alters, and re-broadcasts messages.
- Confidentiality. Unauthorized parties should not be able to infer the content of messages.
- Ease of use. Finally, taking into account the diversity of sensor networks users, TinySec should not be difficult to use. We hope to provide a communication stack that provides the above three goals which is no more difficult to use than the traditional, non-security aware communication stack.

TinySec works both in the TOSSIM simulator as well as on the Mica and Mica2 motes.

2 Installation

TinySec is now included with the standard TinyOS distribution. Building TinySec enabled applications uses the `mote-key` script. Make sure this script is in your `PATH`.

2.1 Testing Your Installation

The TestTinySec application can be used to check your TinySec installation. It periodically sends a data packet using TinySec. SecureTOSBase is the TinySec-aware analogue to TOSBase and will toggle red when it receives a valid packet.

You should program two motes, one with TestTinySec and one SecureTOSBase

```
$
$ cd nest/tinyos-1.x/apps/TestTinySec
$ make mica2 install
$ # Now install the second mote
$ cd nest/tinyos-1.x/apps/SecureTOSBase
$ make mica2 install
```

TinySec is properly working if the SecureTOSBase mote toggles its red LED periodically. This indicates that the packet was transmitted with authentication enabled.

If either application fails to build, then check that the "mote-key" script is in your `PATH`.

2.2 Writing Applications

Enabling TinySec for your applications should be a straightforward process. In general, application code does not need to change[1]. If your Makefile includes Makerules from the `tinyos-1.x/apps` directory, you only need to add `TINYSEC=true` to your application's Makefile in order to enable TinySec. This can also be be done from the command line when invoking make: e.g., `make mica2 TINYSEC=true`.

By default, TinySec will authenticate all messages, but encryption is turned off. TinySec allows you to dynamically alter the combination of security mechanisms for your application with the `TinySecMode` interface. The `TinySecC` component exports this interface. There are two commands in the `TinySecMode` interface for setting the transmit and receive mode:

```
command result_t setTransmitMode(uint8_t mode);
command result_t setReceiveMode(uint8_t mode);
```

`setTransmitMode` takes one of three values as its argument:

```
TINYSEC_AUTH_ONLY
TINYSEC_ENCRYPT_AND_AUTH
TINYSEC_DISABLED
```

Similarly, `setReceiveMode` takes one of three values as its argument:

```
TINYSEC_RECEIVE_AUTHENTICATED
TINYSEC_RECEIVE_CRC
TINYSEC_RECEIVE_ANY
```

The default is `TINYSEC_AUTH_ONLY` for sending and `TINYSEC_RECEIVE_AUTHENTICATED` for receiving. `TINYSEC_ENCRYPT_AND_AUTH` sends encrypted and authenticated messages, and `TINYSEC_DISABLED` sends messages with the standard TinyOS radio stack (CRC only, with no encryption or authentication).

When the receiver is in `TINYSEC_RECEIVE_AUTHENTICATED` mode, it will only accept messages from a sender in `TINYSEC_AUTH_ONLY` mode or `TINYSEC_ENCRYPT_AND_AUTH` mode. A receiver in `TINYSEC_RECEIVE_CRC` mode will only accept messages from a sender in `TINYSEC_DISABLED` mode, and a receiver in `TINYSEC_RECEIVE_ANY` mode will accept messages from senders in any mode. The application layer can check which mode a message was received in by inspecting the `receiveSecurityMode` in the corresponding `TOS_Msg`. This interface is subject to race conditions.

[1]Except for applications using the group field. See Section 2.6 about deprecation of the group field in the packet header.

2.3 SecureTOSBase

Use the `SecureTOSBase` application to interface your PC with a network of motes enabled with TinySec.

`SecureTOSBase` will only accept messages that that have been sent with a MAC; thus it will *not* receive messages sent with the old radio stack or with TinySec disabled. You will need to modify the `SecureTOSBase` application if you would like to receive both authenticated and unauthenticated messages.

2.4 Key Management

When using TinySec, you must be aware of keys and the key-file. Each Mica mote can only communicate with other motes that have been programmed with the same key. The key is currently set in a given program at build time. Without any extra arguments to the normal build process, the default key-file and default key will be used. A default key-file will be created for you the first time TinySec is used and stored in the file at `~/.tinyos_keyfile`. It looks like:

```
# TinySec Keyfile. By default, the first key will be used.
# You can import other keys by appending them to the file.

default 6D524D67F24F178B0A69933FDD6C6F7B
```

Note that the your key value will not be the same as listed above. Each line lists a key name and the key value. When you invoke `make mica2`, the first key in the default key-file will be installed.

This means that by default, if you install a program onto one mote from your laptop, and install a program onto another mote from your desktop, they will not be able to interoperate. This is because they will be using different keys. Thus, you'll need to perform one of the following:

- Use the same key-file on both computers
- Copy the key-file from your laptop to the desktop, renaming the file to "laptop-keyfile". Then, when building on the desktop, use the new keyfile whenever you wish to create motes that interoperate with motes programmed from the laptop:

  ```
  make mica2 KEYFILE=laptop-keyfile
  ```

- Copy the line from the keyfile which reads "default 6D524..." from the laptop to the desktop keyfile (.keyfile). In addition, rename the key label from "default" to "laptop". Then, when building on the desktop, use the new keyfile whenever you wish to create motes that interoperate with motes programmed from the laptop:

  ```
  make mica2 KEYNAME=laptop
  ```

2.5 Updating Keys

The `TinySecControl` interface exported by the `TinySecC` component enables you update TinySec keys and query and reset the initialization vector (IV). The `TinySecControl` interface has six commands:

```
command result_t updateMACKey(uint8_t * MACKey);
command result_t getMACKey(uint8_t * result);
command result_t updateEncryptionKey(uint8_t * encryptionKey);
```

```
    command result_t getEncryptionKey(uint8_t * result);
    command result_t resetIV();
    command result_t getIV(uint8_t * result)
```

updateMACKey() and updateEncryptionKey() update the corresponding key, and take a pointer to an array of TINYSEC_KEYSIZE bytes. These commands return SUCCESS if the key update is successful. These commands will return FAIL if the key update is attempted while any crypto operation is executing in TinySec. This is the simplest solution to ensure atomicity of crypto operations with respect to the TinySec keys. The caller of these commands must check their return values, and try again if they return FAIL.

getMACKey() and getEncryptionKey() return the corresponding key into the array referenced by the input parameter result. result must reference a buffer of size at least TINYSEC_KEYSIZE bytes.

resetIV() resets the counter portion of the IV to 0. This will most likely be used in conjunction with updating the encryption key. As in the key update commands, resetIV() will return FAIL if it is called while any crypto operation is executing in TinySec. getIV() returns the current value of the IV into the array referenced by the input parameter result. result must reference a buffer of size at least TINYSEC_IV_LENGTH bytes.

2.6 Groups

The concept of "groups" does not exist under TinySec. Groups are a means to provide namespaces for destination addresses when sending packets. Each mote belongs to one of 255 different groups. The group byte is transmitted as a part of every packet. A packet is accepted only if the sender and receiver are in the same group.

This functionality is subsumed by using keys under TinySec. Using TinySec, a message will only be accepted if the sender and receiver use the same key. This is enforced cryptographically. However, if a received message passes the MAC, the TOS_Msg received at the application layer will show the group that the receiving mote was programmed with.

3 Under the Covers

We've created replacements for the TinyOS radio stacks that incorporate TinySec. These files are located in tinyos-1.x/tos/platform/{mica2,mica,pc}/TinySec. The remaining TinySec components are located in tinyos-1.x/tos/lib/TinySec. See http://www.cs.berkeley.edu/~nks/tinysec for more information.