TinySec: User Manual

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

September 12, 2003

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

Building TinySec enabled applications uses the `mote-key` script. After conducting a `make install` operation in the `apps` directory, the script should reside in `/usr/local/bin`. You will need to add this directory your shell's path before compiling a TinySec enabled application, if it is not already there.

See Section 2.2 for instructions on how to TinySec enable an existing application.

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.

We've created a new component called `SecureGenericComm` that is meant to replace the existing radio stack `GenericComm`. `SecureGenericComm` exports three different send interfaces:

1. `SendMsg` This sends a message using a cryptographically strong MAC. There is no encryption provided. The recipient must also use the `SecureGenericComm` radio stack.

2. `SendMsgEncryptAndAuth` This sends a message using a cryptographically strong MAC and encrypts the message. The recipient must also use the `SecureGenericComm` radio stack.

3. `SendMsgCRC` Sends a message using a CRC and not encrypted. This interface is not compatible with the existing `GenericComm` radio stack.

Likewise, `SecureGenericComm` exports two different receive interfaces:

1. `ReceiveMsg` Signals messages in which the MAC successfully passed. This interface is meant to provide the receiver a guarantee about the authenticity and integrity of messages received.

   Thus, messages sent from a mote with the `GenericComm` radio stack will not be passed up through this interface since it does not put a MAC on the message.

2. `ReceiveMsgNoAuth` Signals messages in which the CRC successefully passed. This interface does not provice the receiver with any guarantees about the authenticity of the message received. Note that messages sent over `GenericComm` are not compatible with this interface and will not be received successfully.

Simply replacing `GenericComm` with `SecureGenericComm` will therefore provide authenticity guarantees for the application but not confidentiality guarantees. To enable encryption, the sender must use the `SendMsgEncryptAndAuth` interface instead.

In general, application code does not need to change[1].

You will also need to modify your application's Makefile in order to enable the Makefile changes. this can be accomplished by adding `TINYSEC=true` to the application's Makefile or by invoking make: `make mica2 TINYSEC=true`.

[1]Except for applications using the group field. See Section 2.5 about deprecation of the group field in the packet header.

2.3 SecureTOSBase

The `TOSBase` application uses the `GenericComm` radio stack. 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 the `SendMsgCRC` interface. You will need to modify the `SecureTOSBase` application if you would like to receive both authenticated and unathenticated 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 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

See `http://www.cs.berkeley.edu/~nks/tinysec` for more information.