BitVisor 1.1 Reference Manual

Ver. 1.0

2010/12/21
IGEL Co., Ltd

■ Revision History

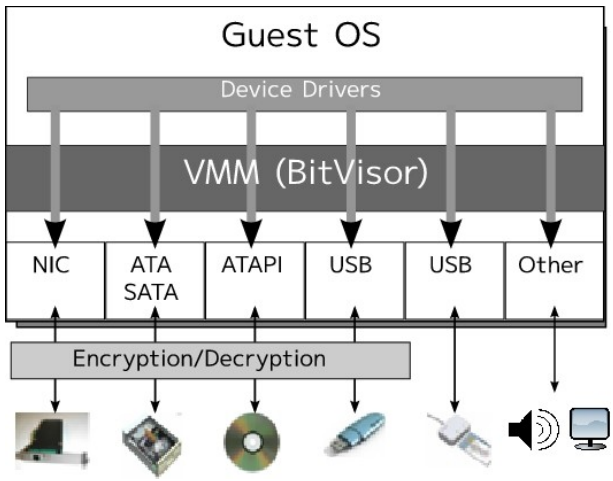
Rev. Date	Revision Comments	Rev. By
	1.0 Initial Draft	DamianHG

Table of Contents

1 Overview

BitVisor is a Virtual Machine Monitor (VMM), designed to enhance the security of computing systems by providing data encryption services for both storage media and network connections. Implemented as a VMM, BitVisor can secure a computer regardless of the operating system (eg. Windows, Linux, etc) and independent of the software configuration being used on the system. BitVisor provides protection from unintentional circulation of information with minimal overhead, allowing secure data transactions at maximum speed.

1.1 Features



- Provides boot up login restriction by password or IC smart card.
- Provides data encryption for HDD, USB storage, CD-RW and other storage media
- Provides automatic and secure network data encryption and access to user specified Virtual Private Networks (VPNs)
- Once installed, BitVisor is transparent to the end user. (i.e. VPN connection can be achieved without the need for a client side application)
- The same configuration can be run regardless of the guest OS used (although different installation methods are required)

BitVisor uses the CPU Virtualization functions to add security enhancements to specific I/O interfaces, while allowing other interfaces to operate normally. This allows for a more efficient virtual machine than one without any hardware support.

2 Requirements

BitVisor has differing requirements for the compilation and the target system (system that will be secured by BitVisor)

2.1 Compilation System

- C-Language compilation environment (required)
  - Fedora 7, Debian GNU/Linux 4.0 recommended
  - For a 64-bit system, a 64-bit compilation environment is necessary
- Access to a Linux kernel tar archive or Internet access (for downloading the archive)
- Details of the specific compilation tools required appears in Chapter 5 Compilation.

2.2 Target System

- A 32-bit or 64-bit Intel® VT or AMD-V™[1] hardware virtualization capable CPU and supporting BIOS.
- 128MB of system memory (in addition to the memory requirements of the guest OS and applications)
- 32-bit guest OS
  - Windows 7, Windows Vista, Windows XP, Fedora 7, Debian GNU/Linux 4.0 recommended
- PS/2 keyboard (required for advanced debug use only)

2.3 Target System supported devices

BitVisor supports the following hardware for stored data encryption and network traffic encryption

- EHCI/UHCI USB Host Controllers
- PATA/SATA/ATAPI disk drives
- Intel PRO 100/1000 series NIC
  - 82566MM Gigabit Network Connection
  - 82566DC Gigabit Network Connection
  - 82567LM Gigabit Network Connection
  - 82572EI Gigabit Ethernet Controller
  - 82562V-2 10/100 Network Connection
- RealTek RTL 8169 NIC
  - RTL8111/8168B PCI Express Gigabit Ethernet Controller
- SCR3310, SCR331FI-NTTCOM[2] IC Card Readers
- RS-232 serial port (for debug)

1 AMD-V™ support is experimental
2 Custom firmware required in order to use contactless smart card feature

3 Downloading

The source code for BitVisor can be found at http://sourceforge.net/projects/bitvisor/

The source code archive file should be extracted to a location with sufficient free disk space. The archive extraction can be performed with the following command:

```
$ tar xjf bitvisor-1.0.1.tar.bz2
```

where *bitvisor-1.0.1.tar.bz2* is the name of the BitVisor source archive file.

Once extracted the following directory structure will have been created.

```
/
+---boot                        #configuration and boot data
|    +---login                  #IC smart card verification
|    +---login-simple           #password verification
|    +---loader                 #boot loader
+---core                        #bitvisor core source files
+---crypto                      #cryptographic routine sources
+---drivers                     #device driver sources
+---idman                       #IC smart card driver sources
+---include                     #common include files
+---storage                     #storage device abstraction
+---vpn                         #vpn connectivity sources
```

For common use of BitVisor, only the /boot subtree is of importance. The other directories are of course essential for development purposes.

4 Configuration

BitVisor configuration **MUST** be completed before BitVisor can be used. The configuration file instructs BitVisor to encrypt the appropriate storage devices and to set up the VPN connection. A configuration file template, `bitvisor.conf.tmpl`, is available in both the `login` and `login-simple/conf` sub-directories of the `/boot` directory in the BitVisor source tree. This file may be copied to `bitvisor.conf` and then edited in order to configure BitVisor.

> **Note:** It is important to note that the configuration information is **compiled** into the BitVisor password login and IC smart card verification files. As a result, these files must be recompiled and reinstalled if the BitVisor configuration changes.

In order to use BitVisor, user verification is necessary. BitVisor supports user verification either by password or by IC smart card. One of these two options must be configured and compiled in order to use BitVisor.

4.1 Password Login Configuration

If a BitVisor secured system will be accessed using a password, the `login-simple` library must be configured.

The `login-simple` version of the `bitvisor.conf.tmpl` configuration file can be found under the `boot/login-simple` sub-directory of the BitVisor source tree.

The default version of this file can be used to initially test a BitVisor installation without enabling any of the security features.

4.2 IC Smart Card Verification Configuration

If a BitVisor secured system will be accessed using an IC smart card, the `login` library must be configured.

The `login` version of the `bitvisor.conf.tmpl` configuration file can be found under the `boot/login/conf` sub-directory of the BitVisor source tree.

The following fields **MUST** be set in order to use this configuration:

```
idman.pkc01File=<IC card authentication certificate file>
idman.randomSeedSize=128
idman.maxPinLen=16
idman.minPinLen=16
idman.authenticationMethod=PKI
```

The *<IC card authentication certificate file>* file should be provided by the Certificate Authority (CA) that has signed the X.509 authentication certificates used on the IC smart card. Additional information on the IC smart card and authentication can be found in Section 8.5 IC Smart Card Data.

This basic configuration can be used to initially test a BitVisor installation without enabling any of the security features.

4.3 BitVisor boot loader

If the BitVisor boot loader will be used (as opposed to GRUB), the following option **MUST** be set in the `bitvisor.conf` configuration file.

```
vmm.boot_active=1
```

This option will instruct BitVisor to automatically chain load the first active (or bootable) partition that it can find on the system instead of relying on GRUB to choose a boot sector to boot.

4.4 Further Configuration

Further information about configuring commonly used features of BitVisor can be found in Chapter 8 Management. A complete list of available configuration commands can be found in Appendix B - BitVisor Configuration. At this point BitVisor compilation and installation can be tested either with the current default configuration, or after more detailed configuration has been completed.

5 Compilation

In order to use BitVisor, the following files must be generated

- Main BitVisor binary
  - bitvisor.elf
- Verification module files (choose one of password or IC verification)
  - module1.bin, module2.bin (password verification files)
  - vmlinux, initrd.img (IC smart card verification files)
- Simple boot loader to load BitVisor at system start (optional)
  - bootloader

In addition to the compilation system requirements defined in Chapter 2 Requirements, the following tools are necessary for compilation of BitVisor components.

- gcc
- binutils
- GNU make

Further information on the gcc compiler can be found at http://gcc.gnu.org/

Further information on GNU make can be found at http://www.gnu.org/software/make/

5.1 Compilation of BitVisor ELF Source

Users who have a pre-compiled binary version of BitVisor may skip to the next section, to proceed with the compilation of their configuration data files.

The following packages are necessary for compilation of the BitVisor binary.

RedHat users

- libusb-dev package
- libssl-dev package

Debian users

- libusb-devel package
- openssl-devel package

Further information on the libusb package can be found at http://www.linux-usb.org/

Further information on the libssl/openssl package can be found at http://www.openssl.org/

The BitVisor binary bitvisor.elf can be generated by

```
$ cd bitvisor-1.0.1
$ make
```

to use the default settings, or by

```
$ cd bitvisor-1.0.1
$ make config
$ make
```

to adjust the defaults before compilation. Details of the compilation configuration settings can be found in Appendix A – Compilation Settings, however, the default values are sufficient for most configurations.

5.2 Compilation of Configuration Data

5.2.1 Password Login Source Compilation

The `login-simple` source can be found under the `boot` sub-directory, and compiled using the `make` command

```
$ cd boot/login-simple
$ make
```

During the compile, when prompted, enter the password to be used when BitVisor is launched at system startup.

```
Enter Password: <my password>
```

Once compilation has been successfully completed, two files, `module1.bin` and `module2.bin`, will have been created. These files will be used during the BitVisor installation process.

5.2.2 IC Smart Card Verification Source Compilation

The `login` source can be found under the `boot` sub-directory, and compiled using the `make` command

```
$ cd boot/login
$ make
```

Once compilation has been successfully completed, two files, `vmlinux` and `initrd.gz`, will have been created. These files will be used during the BitVisor installation process.

5.3 Compilation of BitVisor Boot Loader Source

The BitVisor boot loader provides a method to load BitVisor every time the system is powered up. Alternatively, using a third party boot loader, such as GRUB allows the user to determine whether they want to run BitVisor or not at every system start and choose from several operating systems.

Both of these options can be useful depending on the application.

For users wishing to use the BitVisor boot loader, the `bootloader` source can be found under the `boot/loader` sub-directory, and compiled using the `make` command

```
$ cd boot/loader
$ make
```

Once compilation has been successfully completed the `bootloader` file will have been created. This file will be used during the BitVisor installation process.

6 Installation

Installation of BitVisor can only be performed on a system meeting the requirements for a target system specified in Chapter 2 Requirements. Installation consists primarily of enabling/connecting the necessary hardware and installing a boot loader with the appropriate configuration.

6.1 Enable VT in BIOS

Virtualization Technology (VT) support must be enabled in BIOS in order to use BitVisor. Before attempting to install BitVisor, please ensure that this functionality is enabled in your BIOS configuration.

6.2 Installing a Boot Loader

The possible installation options for BitVisor will depend heavily on the nature of the guest OS being run on the system. Since the installation involves the set up of a boot loader, the type of OS being used will determine which installation methods can be supported. The type of OS supported will be indicated in the installation instructions in each of the following sections

6.2.1 Using GRUB for DOS (Windows ONLY)

The default Windows boot loader, NTLDR, alone is insufficient for loading BitVisor. However, the default boot loader can be used to load a third party boot loader, GRUB4DOS. This boot loader can then be used to launch BitVisor. Installation instructions and the required files for GRUB4DOS can be found on that project's home page, http://sourceforge.net/projects/grub4dos/

Before configuring GRUB4DOS, the required BitVisor binary (bitvisor.elf) and configuration files (module1.bin and module2.bin for password verification and vmlinux and initrd.gz for IC smart card verification) should be copied to a directory on the hard drive of the system that will run BitVisor. It is recommended that the files be copied to an easily identifiable path (eg. /bvboot) to make it easier to apply configuration updates in the future.

When configuring GRUB4DOS, the following should be appended to the end of the menu.lst file that controls the GRUB4DOS boot menu.

```
title BitVisor
    root <device>
    kernel <bvboot>/bitvisor.elf
    module <bvboot>/<module1>
    module <bvboot>/<module2>
```

In this configuration data *<device>* represents the name of the device where the BitVisor files are installed (eg. hd0,0 if the files are on the first partition of the first drive of the system). *<bvboot>* of course represents the path to the BitVisor files. *<module1>* and *<module2>* refer to the configuration files (module1.bin and module2.bin respectively for password login or vmlinux and initrd.gz respectively for IC smart card verification).

6.2.2 Using Legacy GRUB (Linux ONLY)

Users of most Linux distributions can use GRUB for multi-boot support. This section covers the required configuration settings for the Legacy version (up to and including version 0.9x) of the GRUB boot loader.

Before configuring GRUB, the required BitVisor binary (bitvisor.elf) and configuration files (module1.bin and module2.bin for password verification and vmlinux and initrd.gz for IC smart card verification) should be copied the /boot directory on the hard drive of the target PC. The following should then be appended to the end of the menu.lst file that GRUB uses for configuration.

```
title BitVisor
    root <device>
    kernel /boot/bitvisor.elf
    module /boot/<module1>
    module /boot/<module2>
```

The value of *<device>* is usually the same as in the Linux section of the menu.lst file. *<device>* refers to the drive on which BitVisor is installed (often (hd0,1) or something similar). *<module1>* and *<module2>* refer to the configuration files (module1.bin and module2.bin respectively for password login or vmlinux and initrd.gz respectively for IC smart card verification).

6.2.3 Using GRUB2 (Linux ONLY)

Before configuring GRUB2, the required BitVisor binary (bitvisor.elf) and configuration files (module1.bin and module2.bin for password verification and vmlinux and initrd.gz for IC smart card verification) should be copied to the /boot the hard drive of the system that will run BitVisor. The following should then be appended to the end of the grub.cfg file that GRUB2 uses for configuration.

```
menuentry "BitVisor" {
    insmod ext2
    set root=<device>
    multiboot /boot/bitvisor.elf
    module /boot/<module1>
    module /boot/<module2>
}
```

The values of *<device>* is usually the same as in the Linux section of the menu.lst file. *<device>* refers to the drive on which BitVisor is installed (often (hd0,1) or something similar). *<module1>* and *<module2>* refer to the configuration files (module1.bin and module2.bin respectively for password login or vmlinux and initrd.gz respectively for IC smart card verification).

6.2.4 Using BitVisor's Boot Loader (Windows or Linux with custom configuration)

Before installing the BitVisor boot loader a sufficient amount of blank space must be prepared on the target hard drive. The current partitions on the drive must be shrunk to accommodate this space. This can usually be accomplished with a drive management tool, or from the Windows disk management utility. The full BitVisor installation will require about 20MB of installation space, but reserving at least 50MB is recommended.

Once the required space has been reserved in an UNFORMATTED partition, the BitVisor boot loader can be installed. In order to direct the installation script it is important to note the LBA address (or sector address) of the beginning of the unformatted region. This is where the BitVisor data will be installed.

Unlike the GRUB installations, the BitVisor boot loader will load ONLY BitVisor. In order to load the guest OS, the following option **MUST** be set in the `bitvisor.conf` configuration file.

```
vmm.boot_active=1
```

This option will instruct BitVisor to automatically chain load the first active (or bootable) partition that it can find on the system. This has a specific impact on the Linux boot loading procedure (This affects dual boot Linux/Windows systems as well).

> **Linux Customization**
>
> Since the Linux boot loader (GRUB or LILO) is usually installed in the MBR, it cannot be chain loaded by BitVisor. To use Linux with the `vmm.boot_active=1` setting, the Linux stage 1 boot loader must be located in the boot sector (or Volume Boot Record) of a drive instead of the Master Boot Record (the default setting). Details on installing the boot loader to a volume boot sector can usually be found in the documentation of the appropriate boot loader.

Alternatively, since the installation script can only be executed from a Linux command line, a Windows installation of the boot loader also has special requirements.

> **Windows Customization**
>
> In order to install the BitVisor boot loader on a Windows PC access to a working Linux shell is necessary. This can be provided by a Linux boot CD or secondary bootable hard drive (or other storage media). Once the install script and the necessary installation files are all visible from the Linux command prompt, the BitVisor boot loader can be installed. The Linux environment is not necessary after the boot loader installation is complete.

In order to load the BitVisor boot loader on the current machine, execute:

```
$ install.sh
```

from the `boot/loader` directory in the BitVisor source tree.

The command line parameters for the install script are as follows:

```
  usage: ./install.sh [-f] device lba1 lba2 loader elf module1
module2
     -f         first time (do not check existing data)
     device     write to device (ex. /dev/sda)
     lba1       install a boot record to device's lba1
                     (0 for MBR)
     lba2       install images to device's lba2
     loader     /path/to/bootloader
     elf        bitvisor.elf
     module1    login program (vmlinux)
     module2    login data (initrd)
```

The `install.sh` script is used to write the data from the required BitVisor files onto the hard drive and to install the BitVisor boot loader into the Master Boot Record. The command line parameters `loader`, `elf`, `module1` and `module2` specify the paths to the `bootloader`, `bitvisor.elf` and support files (`module1.bin` and `module2.bin` for password login and `vmlinux` and `initrd.gz` for IC smart card login) respectively.

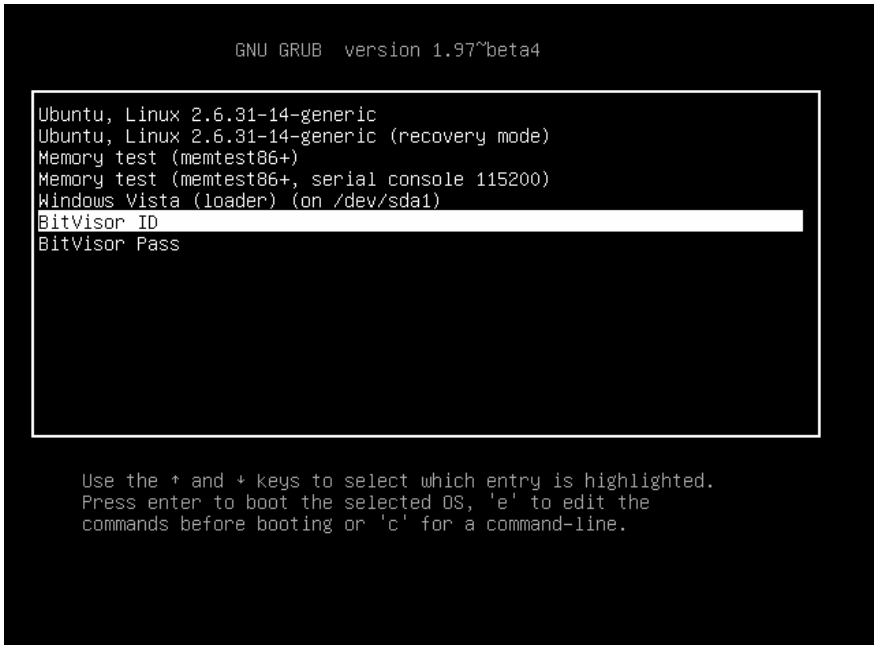
The device parameter specifies which drive the data will be placed on and the `lba1` and `lba2` parameters specify where on the drive the data will reside. lba1 is typically specified as 0 (indicating MBR) so that the boot loader will run at boot up time, while `lba2` should indicate the lba address (or sector address) of the unformatted disk region that will hold the BitVisor data.

The `install.sh` script will overwrite any data at the device address indicated by lba2, destroying any data saved there. Therefore, a check has been implemented to check for over-writable BitVisor data before writing data in an attempt to reduce the possible accidental damage. When being run for the first time, however, this data will not be present and the `-f` flag should be set to force the data to be written to the drive (after the value of lba2 has been double or triple-checked).

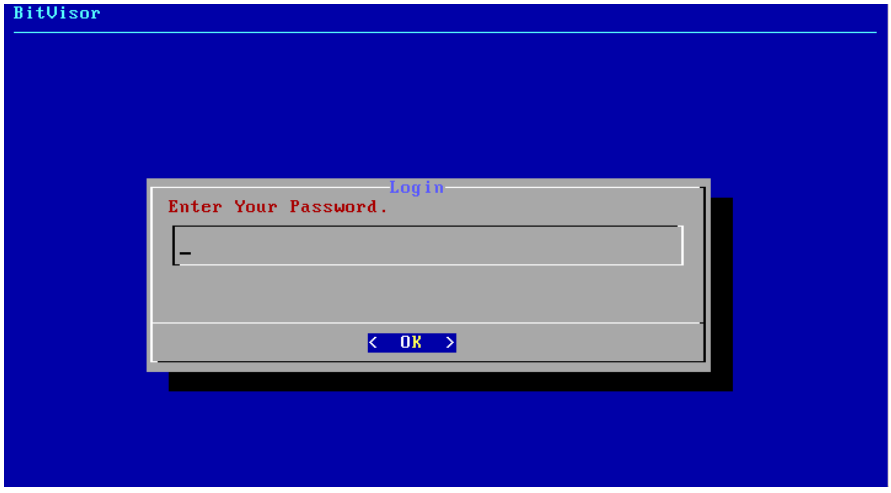
TBD: bootloaderusb

7 Basic Operation

Depending on the type of boot loader used for installation, the initial boot up screen maybe be slightly different (if the BitVisor boot loader is installed, the initial boot up screen will be bypassed, and the login screen will be presented immediately). In general, however, a GRUB boot loader screen will be displayed with an option to load BitVisor.



Once BitVisor is loaded, the login window will appear. Here the login password (specified at compile time of `module1.bin` and `module2.bin`) or IC Smart Card PIN can be entered.



Once BitVisor has been been loaded either the guest OS will automatically boot (if the BitVisor `vmm.boot_active=1` setting appears in the BitVisor configuration file) or the GRUB boot menu will be presented again. If the GRUB menu is presented, the user should select the desired guest OS from the list to continue the system start up procedure.

Your system is now BitVisor secured.

8 Management

8.1 Overview

Management of BitVisor consists of specifying appropriate values for the configuration file, `bitvisor.conf`, and recompiling the necessary support files, either `module1.bin` and `module2.bin` or `vmlinux` and `initrd.gz`, depending on the login configuration. (See Chapter 5 Compilation for compilation details). Use of an IC smart card also requires separate utilities to configure the data embedded on the card. Details of the IC smart card configuration can be found in Section 8.5 IC Smart Card Data.

BitVisor does not necessarily need to be installed on the system being used for configuration/compilation. In fact, the configuration can be compiled on one machine and used to secure several other target computers. This may require some similarity between the target computers (especially with regard to the hard drive configuration), but this is easily achieved, especially if the same drive image is used to initialize all of the computers on a new installation. For further details on any of the configuration settings presented here consult Appendix B - BitVisor Configuration.

8.2 Enabling/Disabling BitVisor features

Before being able to use the various security features of BitVisor, they must be enabled in the `vmm` section of the BitVisor configuration file. Details of this section of the configuration file appear in Section B.4 Virtual Machine Settings.

8.3 Storage Device Encryption

The storage device encryption section of the `bitvisor.conf` configuration file controls how the storage devices will be encrypted. The settings in this section of the configuration file all begin with the keyword `storage`.

In order to encrypt the data on a storage device, the following information is required:

- an encryption key
- the type of device
- the area of the device to encrypt

BitVisor currently supports the AES-XTS encryption with a 256-bit key. The encryption key is generated by the user using any means the user desires. The key can be stored in either a file on disk (for password verification) or on the IC smart card if it is used. If an IC smart card is used, placing the encryption key on the IC smart card is recommended.

The location of the encryption key is specified by

```
storage.encryptionKey<keynum>.place
```

Multiple keys can be specified for encrypting multiple drives. As such, the *<keynum>* index is an integer (starting from zero) used to differentiate the different keys in use.]

For each drive to be encrypted in the system the several fields must be specified. Like the encryption key field, since multiple storage devices may be encrypted, a zero-based integer index is used to specify which device to configure.

```
storage.conf<num>.keyindex
storage.conf<num>.crypto_name
storage.conf<num>.keybits
```

Here the keyindex field corresponds to the <keynum> of the encryption key that the storage device will use. The crypto_name and keybits fields **MUST** be given as aes-xts as 256 respectively as this is the only configuration currently supported.

The specific device to encrypt is given by

```
storage.conf<num>.type
storage.conf<num>.host_id
storage.conf<num>.device_id
```

The type of device can be either ATA, ATAPI, USB, AHCI or AHCI_ATAPI. The host_id and device_id fields specify the specific device. For ATA/AHCI/ATAPI/AHCI_ATAPI devices host_id corresponds to the controller to use (primary = 0, secondary = 1) while device_id corresponds to the device on the control (0 = master, 1 = slave). For USB connected storage devices, these values will change depending on the port used and the order in which the devices are connected. As a result, using a value of -1 meaning accept any value) for the device_id and host_id fields is recommended for USB connected devices.

In order to enable encryption of the device the appropriate BitVisor driver must be enabled in the vmm section of the configuration file.

```
vmm.driver.ata
vmm.driver.usb.uhci
vmm.driver.usb.ehci
```

vmm. driver. ata corresponds to ATA, AHCI, ATAPI and AHCI_ATAPI settings of storage. conf<num>. type while the vmm. driver. usb. uhci/ehci correspond to the USB 1.1 and 2.0 standards respectively.

Once the device is specified, the area of the device to encrypt is given by the LBA (sector) start and end offset on on the drive.

```
storage.conf<num>.lba_low
storage.conf<num>.lba_high
```

8.3.1 Encrypting existing data on a drive

BitVisor encrypts data as it writes to an encrypted drive. Therefore, a newly encrypted drive must be formatted before use. If there is already data on a drive that a user wishes to have encrypted, it is necessary to copy that information from the unencrypted storage device to a BitVisor encrypted device AFTER BitVisor has been installed.

8.4 Setting up an encrypted network connection

BitVisor acts as an interface between the guest OS and the physical network. It does this by creating a virtual network between itself and the guest OS and acting as a gateway between this virtual network and the external physical network (see Figure 1). When configuring BitVisor to set up this configuration, parameters for both of these networks must be specified. In general the virtual network is referred to as the "g*uest* " space while the physical is referred to as the *"host"* space when setting configuration values. The settings in this section of the configuration file all begin with the keyword vpn.

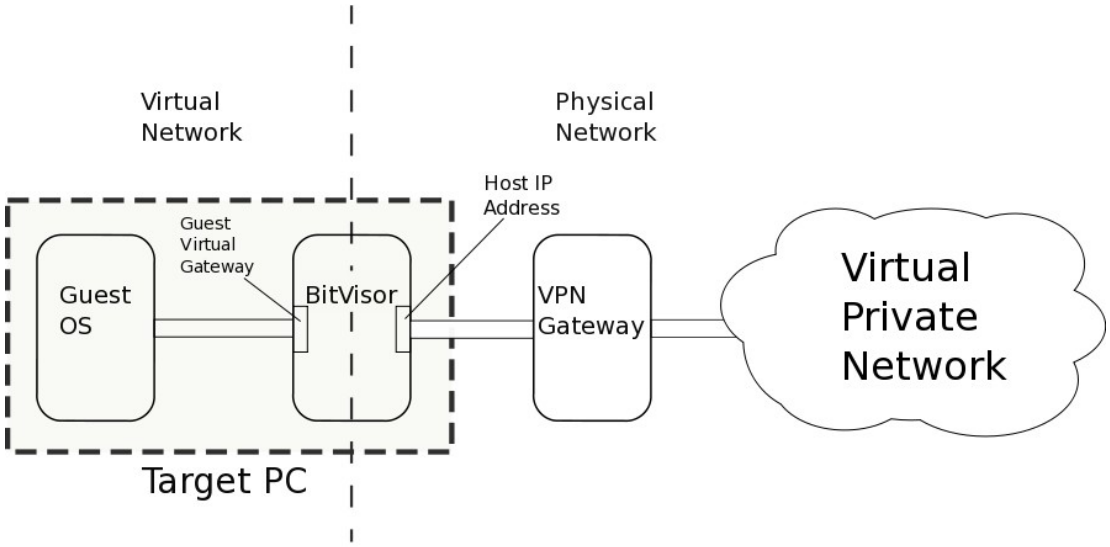


Figure 1: BitVisor Secured Network Connectivity

In order to access the VPN, the appropriate driver for the NIC on the target system must be enabled by setting one of

```
vmm.driver.vpn.PRO100
vmm.driver.vpn.PRO1000
vmm.driver.vpn.RTL8169
```

BitVisor operates in one of three modes for network traffic encryption, controlled by the

```
vpn.mode
```

configuration setting. The possible values of L2Trans and L3Trans correspond to transparent layer 2 and layer 3 modes, used for testing. In the L3IPsec mode, BitVisor will create the IPSec connection for data transmission.

BitVisor supports both IPv4 and IPv6 IPSec connections, with different settings for both (the V4 or V6 in the setting name will indicate to which type of connection the setting applies). These may be enabled/disabled separately by setting

```
vpn.bindV4(V6)
```

to a value of TRUE.

8.4.1 Virtual Network Details

BitVisor creates a virtual network between the guest OS and itself. On this network BitVisor acts as a DNS(RA) server, configuring the guest OS with the parameters specified by the following settings:

```
vpn.guestIpAddressV4(vpn.guestIpAddressPrefixV6)
vpn.guestIpSubnetV4(vpn.guestIpAddressSubnetV6)
vpn.guestMTU(V6)
vpn.guestVirtualGatewayIpAddressV4(V6)
```

Since this is a virtual network these settings can be set fairly arbitrarily, with the one exception that the vpn.guestVirtualGatewayIpAddressV4(V6) **MUST** be set to be on the same subnet as the vpn.guestIpAddressV4(V6). (Otherwise the guest OS will never be able to access the physical network)

The physical interface must also be statically configured by the BitVisor configuration file. This is because BitVisor does not contain a DHCP or RA client for auto-configuration. The physical interface can be configured by setting

```
vpn.hostIpAddresssV4(V6)
vpn.hostIpSubnetV4(vpn.hostIpAddressSubnetV6)
vpn.hostMtuV4(V6)
vpn.hostIpDefaultGatewayV4(V6)
```

to appropriate values.

In order to connect to the VPN,

```
vpn.vpnGatewayAddressV4(V6)
```

must also be set. Once the vpn settings are properly configured, the packets sent from the guest OS will be forwarded via the following steps:

1. Guest OS forwards packets to its default gateway – vpn. guestVirtualGatewayIpAddressV4(V6)
2. BitVisor encrypts the packets and attempts to forward them to vpn. vpnGatewayAddressV4(V6)
3. If vpn. vpnGatewayAddressV4(V6) is on a different subnet from vpn. hostIpAddresssV4(V6), BitVisor will attempt to access it through vpn. hostIpDefaultGatewayV4(V6).
4. Packets are decrypted at vpn. hostIpDefaultGatewayV4(V6) where they continue on to their final destination.

8.4.2 VPN Connection Details

BitVisor supports IPSec tunnel connections using ESP encryption. Transport connections are not supported by BitVisor. Using ISAKMP, BitVisor will negotiate encryption keys in two phases in order to set up an IPSec connection.

The Phase I authorization can be provided by either pre-shared key or by an identity

certificate as specified by

```
vpn.vpnAuthMethodV4(V6)
```

where the value of "Password" denotes pre-shared key authorization, while "Cert" denotes authorization using X.509 certificates. The associated settings for pre-shared key authorization are

```
vpn.vpnPasswordV4(V6)
vpn.vpnIdStringV4(V6)
```

while those for X.509 certificate authentication are

```
vpn.vpnCertFileV4(V6)
vpn.vpnCaCertFileV4(V6)
vpn.vpnRsaKeyFileV4(V6)
```

vpn.vpnCertFileV4(V6) is the name of a file containing an X.509 certificate identifying the target PC. This file contains the public key that will be used for verification on the remote VPN server. vpn.vpnRsaKeyFileV4(V6) is the name of the file containing the private key that matches this public key. On the other hand , vpn.vpnCaCertFileV4(V6) is the name of the file containing a CA certificate that can be used to veify the ***remote VPN server's*** X.509 certificate. In this way both the remote VPN server and the target PC can verify the identity of the other party. Section 8.5.1 Authentication Certificates discusses authentication certificates as they apply to IC smart card authentication, but the general concepts are the same as for VPN access.

The authorization data can alternatively be supplied on an IC smart card by specifying "Password-IC" or "Cert-IC" as the vpn.vpnAuthMethodV4(V6). (Note: Since only the vpn.vpnPasswordV4(V6) value is stored in the IC smart card, vpn.vpnIdStringV4(V6) must be specified in the configuration file even when vpn.vpnAuthMethodV4(V6)=Password-IC.)

The remaining IKE setting settings for both Phase I and Phase II can be found in Table 2: VPN IPv4 Settings and Table 3: VPN IPv6 Settings in Appendix B - BitVisor Configuration.

> During the IKE phase I authorization, BitVisor supports only Diffie–Hellman group 2 (mod1024p)

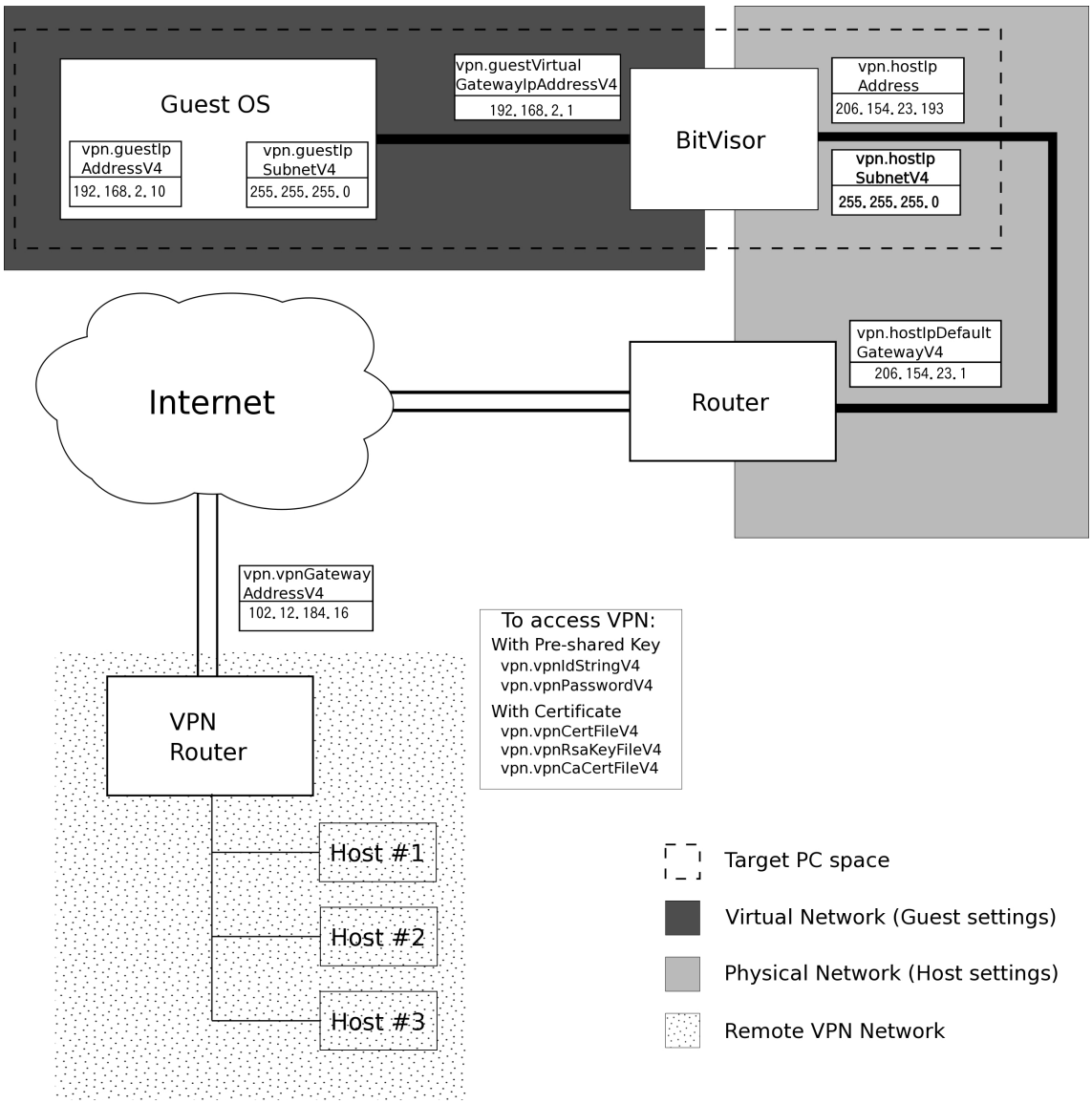
Once the connection has been established, BitVisor will periodically PING an IP address to test the integrity of the secured tunnel connection. The PING target should be set to an IP address within the VPN for proper operation. The settings controlling this function are

```
vpn.vpnPingTargetV4(V6)
vpn.vpnPingIntervalV4(V6)
vpn.vpnPingMsgSizeV4(V6)
```

If vpn.vpnPingTargetV4(V6) is omitted, no PING tests will be performed.

8.4.3 Example

The following IPv4 example shows how a Target PC might be connected to a VPN through the Internet, and how to configure BitVisor to accomplish this. The numeric values are example values only, but they show which configuration values need to be assigned to make a connection successfully.



8.5 IC Smart Card Data

BitVisor supports up to 3 identity certificates to be managed by IC smart card. The IC smart card should contain both an X.509 authentication certificate and a private key, but only the certificate should be readable from the card reader interface. All IC smart card data, should also be protected by a PIN number.

The IC smart card used by BitVisor contains:

Accessible Memory

- up to 3 authentication certificates
- storage media encryption/decryption key
- Pre-shared key for VPN access

Inaccessible Memory

- up to 3 private keys that match the authentication certificate public keys

Functionality

- built-in ability to sign challenge data with a private key

IC smart card verification occurs in two steps:

1. verification of the authentication certificate
2. verification of the IC smart card

8.5.1 Authentication Certificates

The authentication certificate stored on the IC smart card must be *signed* by a Certificate Authority (CA). This signature is verified by BitVisor using a CA certificate file, specified in the BitVisor configuration file setting

```
idman.pkc0xFile
```

where x denotes which of the three certificates the CA certificate applies(possible values of 1, 2, or 3). This CA certificate file should be provided by whatever CA has signed the IC smart card authentication certificates.

Once the authentication certificate (which contains a public key) is verified, the verification of the IC smart card itself is conducted. Random challenge data is presented to the card, which encrypts the data with its private key (that matches the public key in the authentication certificate) and returns it to BitVisor. BitVisor will then decrypt the data with the public key, verifying the identity of the IC smart card.

While up to 3 authentication certificates are supported, at this time, only the first two slots are being used. The first certificate is used for identity verification at system boot time (when the IC smart card PIN is entered). The second certificate is used for VPN authentication.

8.5.2 Screen Lock Functionality (Windows ONLY)

With the help of an external application, BitVisor can supply a screen locking function, tied to the presence of an IC smart card in a card reader. While the card is in the reader the system will function as normal, but when it is removed the mouse, keyboard and screen will all be locked until the IC smart card is returned to the reader.

To configure BitVisor to use this feature, in addition to any other IC smart card configuration settings, the following setting **MUST** be set in the configuration file.

```
vmm.iccard.status=1
```

In order to compile the helper application a Windows targeted version of `gcc` is necessary. Using the `MinGW` libraries, available for `gcc` in both Linux and Windows enviroments, the helper application can be built. This tool is available as a package for many popular Linux distributions, as a Windows execuable and as source code.

Notes for compiling from source files:

- Souce code is available at `http://sourceforge.net/projects/mingw`
- When compiling the source code, the required include files and libraries downloaded from the `MinGW` repositories are referenced when creating Windows targeted versions of `gcc` and `binutils`.
- The `runtime` and `w32api` libraries contain both the header files necessary to build the `gcc` cross-compiler and the source for the libraries that the cross-compiler will use to link other programs. As a result these libraries must also be compiled before the compiler can be used.

The souce code for the helper application can be found as a comment towards the bottom of the `vmmcall_iccard.c` file in the `/core` subdirectory of the BitVisor source code tree.

This code should be copied into a new file and compiled, using a Windows targeted `gcc` compiler with the following command:

```
<compiler name> -mwindows <source file> -o locker.exe
```

The resulting file, `locker.exe`, should be executed whenever the locking functionality is necessary. It may be useful to add a shortcut to `locker.exe` to the Startup directory of the Start Menu/Programs so that it will execute whenever Windows is loaded..

9 Advanced Operation

Advanced users of BitVisor may use the debug interfaces to diagnose configuration problems or other errors. BitVisor supports a command line debug interface, accessible from a command line terminal on the target PC.

9.1 Required configuration settings

In order to access the BitVisor debug interface, one or both of the following configuration file options must be set to "1".

```
vmm.shell
vmm.dbgsh
```

vmm.shell allows access to the debug interface when an error occurs that causes BitVisor to suspend the system (i.e. a "panic" event). vmm.dbgsh, on the other hand, will allow access to the debug interface via a command line tool whenever BitVisor is active.

9.2 Command line tool source code

The source for the command line debug interface tool can be found as a comment in the source code file vmmcall_dbgsh.c in the core/ directory of the BitVisor source tree. Once compiled (simply by using gcc), the tool will be able to access the BitVisor debug interface when run from the command line on the target PC. A full command list for the debug interface can be obtained by typing "help" at the command line, but the most useful command for users is the "log" command. which displays the BitVisor log.

9.3 Access via RS-232

The BitVisor logging output can be redirected to a serial port, where it can be read from a secondary computer. The interface can also be used in order to input commands to the BitVisor debug interface in the event that a BitVisor panic occurs on the target PC.

In order to enable this mode the main BitVisor binary, bitvisor.elf, must be compiled with the

```
CONFIG_TTY_SERIAL
```

option set.

With this option set, BitVisor will redirect its input and output to the first serial port, using the following settings:

BPS rate: 115200 bps

data bits: 8

parity bits: none

stop bits: 1

9.4 PS/2 Keyboard Debug commands

Some special commands are available to users with access to a PS/2 keyboard and PS/2 input port on the target PC. These commands must have support enabled both when the bitvisor.elf file and the configuration files are compiled. The associated configuration

settings are:

bitvisor.elf compile

```
CONFIG_PS2KBD_F11PANIC
CONFIG_PS2KBD_F12MSG
```

configuration file

```
vmm.f11panic
vmm.f12msg
```

As the setting names suggest, pressing the F11 key will cause the BitVisor VMM to issue a panic and stop the system operation. On the other hand, pressing the F12 key will instruct BitVisor to print the following test message:

“F12 pressed.”.

Appendix A - Compilation Settings

Setting	Def	Comment	Notes
CONFIG_64	1	Compile a 64-bit version of BitVisor	If the system does not support 64-bit compilation via the `gcc -m64` flag, the default value will be “0”
CONFIG_DEBUG_GDB	0	Enable remote GDB debug support (32-bit only)	Unsupported and unstable
CONFIG_TTY_SERIAL	0	BitVisor outputs status updates to serial port (COM1)	
CONFIG_TTY_PRO1000	1	BitVisor outputs status updates to LAN	Dependency: CONFIG_VPN_PRO1000
CONFIG_CPU_MMU_SPT_1	0	Shadow Type 1 (slow)	Choose one
CONFIG_CPU_MMU_SPT_2	0	Shadow Type 2 (faster)	
CONFIG_CPU_MMU_SPT_3	1	Shadow Type 3 (fastest)	
CONFIG_CPU_MMU_SPT_USE_PAE	1	Shadow Page Table uses PAE	
CONFIG_PS2KBD_F11PANIC	0	Cause a BitVisor panic when F11 is pressed (PS/2 keyboard only)	
CONFIG_PS2KBD_F12MSG	1	Print a message to the log when F12 is pressed (PS/2 keyboard only)	
CONFIG_DBGSH	1	Enable debug shell access from the guest OS	
CONFIG_STATUS	0	Make BitVisor status available from the guest OS via a third party guest OS space interface tool	Third party interface tool required
CONFIG_LOG_TO_GUEST	0	Make BitVisor log available from the guest OS via a third party guest OS space interface tool	Third party interface tool required
CONFIG_ATA_DRIVER	1	Enable ATA storage device driver	
CONFIG_STORAGE_ENC	1	Enable storage encryption	MUST be enabled
CONFIG_CRYPTO_VPN	1	Enable network encryption	MUST be enabled
CONFIG_USB_DRIVER	1	Enable USB storage device driver	
CONFIG_SHADOW_UHCI	1	Shadow UHCI (USB1.1) transfers	Dependency: CONFIG_USB_DRIVER

Setting	Def	Comment	Notes
CONFIG_SHADOW_EHCI	1	Shadow EHCI (USB2.0) transfers	Dependency: CONFIG_ SHADOW_UHCI
CONFIG_HANDLE_USBMSC	1	Enable support for USB Mass Storage Device class devices	Dependency: CONFIG_ SHADOW_UHCI
CONFIG_HANDLE_USBHUB	1	Enable support for USB Hub class devices	Dependency: CONFIG_ SHADOW_UHCI
CONFIG_CONCEAL_USBCC ID	1	Conceal USB CCID (Smart Card) devices from the guest OS	Dependency: CONFIG_ SHADOW_UHCI
CONFIG_PS2KBD_F10USB	0	Run a test for USB CCID devices when F10 is pressed	Dependency: CONFIG_ SHADOW_UHCI
CONFIG_PS2KBD_F12USB	0	Dump the EHCI asynchronous list when F12 is pressed	Dependency: CONFIG_ SHADOW_UHCI
CONFIG_IEEE1394_ CONCEALER	1	Conceal OHCI IEEE1394 devices from the guest OS	
CONFIG_FWDBG	0	BitVisor outputs status updates to a Firewire (IEEE 1394) device	
CONFIG_ACPI_DSDT	1	Enable ACPI function overrides	
CONFIG_DISABLE_SLEEP	1	Disable ACPI S2 and S3 sleep states	Dependency: CONFIG_ACPI_ DSDT
CONFIG_ENABLE_ASSERT	1	Enable checking assertion failure	
CONFIG_DEBUG_ATA	0	Enable ATA driver debug interface	Dependency: CONFIG_ATA_ DRIVER
CONFIG_SELECT_AES_ GLADMAN	0	Use Gladman AES assembly language code for data encryption	Not officially supported
CONFIG_CARDSTATUS	1	Cause BitVisor to issue a panic if the IC card is removed from the reader	Dependency: CONFIG_ IDMAN
CONFIG_IDMAN	1	Enable IC smart card support	MUST be enabled
CONFIG_VPN_PRO100	1	Enable secure network connection on Intel PRO100 NIC	
CONFIG_VPN_PRO1000	1	Enable secure network connection on Intel PRO1000 NIC	
CONFIG_VPN_RTL8169	0	Enable secure network connection	

Setting	Def	Comment	Notes
		on Realtek RTL8169 NIC	
CONFIG_VPN_VE	1	Enable Virtual Ethernet driver (requires a third party guest OS driver)	
CONFIG_VTD_TRANS	0	Enable Intel VT-d support for increased security	Experimental
CONFIG_STORAGE_PD	0	Add increased security on the storage device interface at the cost of some speed	
CONFIG_IDMAN_PD	0	Add increased security on the IC smart card interface at the cost of some speed	
CONFIG_VPN_PD	0	Add increased security on the VPN interface at the cost of some speed	
CONFIG_DISABLE_TCG_ BIOS	1	Disable TCG BIOS extensions.	
CONFIG_ACPI_TIME_ SOURCE	1	Use ACPI PM Timer as the time source	

Table 1: Compilation Settings

Appendix B - BitVisor Configuration

The BitVisor configuration file instructs BitVisor as to which storage devices to encrypt and how to set up the VPN connection. The file must be configured properly before the BitVisor password dialog and/or IC smart card interface can be compiled/configured.

The configuration file fields are given in Table B.1. The "`Default`" column lists the default values listed in the `bitvisor.conf.tmpl` file. When there are differences between the `login` and `login-simple` versions of this file, they are indicated. No default values are listed here for IP addresses, text fields or filenames, since these are either explicitly specified by the specifics of the user configuration, or completely arbitrary.

All entries with a "○" in the "Omit" column can be omitted if the default value is sufficient for the configuration. Only entries in the VPN Settings can be omitted in this manner. In other sections the settings can be omitted if the functionality is not needed (i.e. if no storage encryption is required, the entire storage section may be omitted).

B.1 VPN Settings

Setting Name	Possible Values	Description	Default	Omit
vpn.mode	L2Trans	Transmit VPN packets on TCP/IP Layer 2 (bridge)	L3IPsec	
	L3Trans	Transmit VPN packets with no encryption on Layer 3		
	L3IPsec	IPSec encryption enabled		
vpn.virtualGateway MacAddress	*48bit MAC address*	Mac address of the virtual network gateway that the guest OS will connect to. This value is only used between the guest OS and BitVisor and so can be set arbitrarily	00-88-88-88-88-88	
vpn.bindV4	TRUE	Enable/Disable Ipv4 functionality of VPN	TRUE	
	FALSE			
vpn.guestIp AddressV4	*Ipv4 address range*	IP4 address to be assigned to the guest OS by the BitVisor DHCP server		
vpn.guestIp SubnetV4	*IPv4 subnet range*	IP4 subnet mask to be assigned to the guest OS by the BitVisor DHCP server		
vpn.guestMTU		MTU to be used by guest OS	1400	○
vpn.guestVirtual GatewayIp	*IPv4 address range*	Default gateway setting that the guest OS will	broadcast address - 1	○

AddressV4		receive from the BitVisor DHCP server		
vpn.dhcpV4	TRUE	Enable DHCPv4 server facing the guest OS	TRUE	
	FALSE			
vpn.dhcpLease Expires		DHCP lease expiry in seconds	3600	○
vpn.dhcpDnsV4	*IPv4 address range*	DNS server that guest OS will be given		○
vpn.dhcpDomain V4		Domain name that guest OS will be given		
vpn.adjustTcpMss V4		Adjust the TCP Maximum Segment Size for the VPN connection. Can only be used to **reduce** the MSS.	1240	
vpn.hostIpAddress V4	*IPv4 address range*	The static values that BitVisor will use to interface with the VPN. Currently the VPN side interface to BitVisor does not include a DHCP client, so static values must be used		
vpn.hostIpSubnet V4	*IPv4 address range*			
vpn.hostMtuV4			1500	○
vpn.hostIpDefault GatewayV4	*IPv4 address range*			
vpn.optionV4Arp Expires		Length of time before ARP entries expire	60	○
vpn.optionV4Arp DontUpdate Expires	FALSE	Update the ARP expiry timer on successful ARP cache access	FALSE (login-simple) TRUE (login)	○
	TRUE	Do not update ARP expiry timer when a successful ARP cache access is performed		
The remaining setting maybe be omitted when not using Mode=L3IPsec and BindV4=TRUE				
vpn.vpnGateway AddressV4	*IPv4 address range*	The remote host to act as the far end of the VPN. The VPN gateway server that the client (guest OS/BitVisor) wishes to connect to		
vpn.vpnAuth MethodV4	Password	Use a pre-shared key to authenticate the host/user on the VPN	Password	
	Cert	Use an X.509 authentication certificate to		

		access the VPN		
	Password-IC	Use a pre-shared key stored on an IC smart card (recommended if IC smart card is used)		
	Cert-IC	Use a X.509 authentication certificate stored on an IC smart card (recommended if IC smart card is used) (vmm.iccard.enable MUST be 1)		
vpn.vpnPassword V4	*ASCII string*	Pre-shared key for user authentication (for binary values, enter in hexadecimal, prefixed with “0x”)		
vpn.vpnIdStringV4	*ASCII string*	VPN ID (FQDN or username) for use with vpnPasswordV4		
vpn.vpnCertFileV4		Target PC's certificate file to use for X.509 authentication for VPN access. File Format: DER		
vpn.vpnCaCertFile V4		VPN (or other party's) CA certificate file for X.509 authentication on the VPN File Format: DER		
vpn.vpnRsaKeyFile V4		Target PC's RSA private key for X.509 authentication on the VPN File Format: DER		
vpn.vpnSpecify IssuerV4	TRUE	Request that the VPN (or other party) use a certificate signed by the same CA as BitVisor's certificate via IKE Certificate Request payload	FALSE	
	FALSE	Do not specify any CA in the IKE Certificate Request		
vpn.vpnPhase1 CryptoV4	DES	Use Data Encryption Standard encryption	3DES	
	3DES	Use Triple DES encryption		

vpn.vpnPhase1 HashV4	SHA-1	Security hash function to use SHA-1 is the only possible value	SHA-1	
vpn.vpnPhase1Life SecondsV4		ISAKMP SA lifetime (in seconds)	7200	○
vpn.vpnPhase1Life KilobytesV4	0	Do not specify ISAKMP SA lifetime data limit	0	○
	>0	ISAKMP SA lifetime (in kilobytes)		
vpn.vpnWaitPhase2 BlankSpanV4		Number of milliseconds to wait between the end ISAKMP Phase 1 and the start of Phase 2	0	
vpn.vpnPhase2 CryptoV4	DES	Use Data Encryption Standard encryption	3DES	
	3DES	Use Triple DES encryption		
vpn.vpnPhase2 HashV4	SHA-1	Security hash function to use SHA-1 is the only possible value	SHA-1	
vpn.vpnPhase2Life SecondsV4		IPSec SA lifetime (in seconds)	7200	○
vpn.vpnPhase2Life KilobytesV4	0	Do not specify IPSec SA lifetime data limit	0	○
	>0	IPSec SA lifetime (in kilobytes)		
vpn.vpnConnect TimeoutV4		Connection request timeout. The number of seconds to wait after issuing a connection request and declaring a failed attempt	5	○
vpn.vpnIdle TimeoutV4		Connection idle timeout. Number of seconds to allow the established VPN connection to be idle before automatically resetting the connection.	300	○
vpn.vpnPing TargetV4	*IPv4 address range*	The address of a remote host within the established VPN. Periodic PINGs will be sent to this host to determine if the VPN is		○

		still active. If this entry is omitted, PINGs are not sent		
vpn.vpnPing IntervalV4		Length of time between PINGs to the host specified by vpn.vpnPingTargetV4	12	○
vpn.vpnPingMsg SizeV4		Length of PING message to send to vpn.vpnPingTargetV4	32	○

Table 2: VPN IPv4 Settings

Setting Name	Possible Values	Description	Default	Omit
vpn.bindV6	TRUE FALSE	Enable/Disable Ipv4 functionality of VPN	FALSE	
vpn.guestIp AddressPrefixV6	*IPv6 address range*	Address portion of the IPv6 address that the guest OS will be given by BitVisor		
vpn.guestIp AddressSubnetV6	0-128	Length of vpn.guestIpAddressPrefix V6 (bits)	64	
vpn.guestMtuV6		The MTU that the guest OS will be told to adhere to.	1400	○
vpn.guestVirtual GatewayIp AddressV6		Default gateway setting that the guest OS will receive from the BitVisor DHCP server	automatically generated from the EUI-64 of the interface	○
vpn.raV6	TRUE FALSE	Enable/Disable sending routing advertisement messages from BitVisor to the guest OS	TRUE	
vpn.raLifetimeV6		Number of seconds after the last RA is received from BitVisor before the guest OS will consider the BitVisor side router to have failed	3600	○
vpn.raDnsV6	*IPv6 address range*	DNSv6 server to be used by the guest OS		○
vpn.hostIp	*IPv6 address*	The static values that		

Setting Name	Possible Values	Description	Default	Omit
AddressV6	*range*	BitVisor will use to interface with the VPN. Currently the VPN side interface to BitVisor does not include a DHCP client, so static values must be used. vpn.hostIpAddressSubnetV6 specifies the IPv6 prefix length of vpn.hostIpAddressV6		
vpn.hostIp AddressSubnetV6	*IPv6 address range*		64	
vpn.hostMtuV6			1500	○
vpn.hostIpDefault GatewayV6	*IPv6 address range*			
vpn.optionV6 NeighborExpires		Neighbor entry cache expiry length in seconds	60	
The remaining setting maybe be omitted when not using Mode=L3IPsec and BindV6=TRUE				
vpn.vpnGateway AddressV6	*IPv6 address range*	IPSec VPN Gateway IP address		
vpn.vpnAuth MethodV6	Password	Use a pre-shared key to authenticate the host/user on the VPN	Password (login-simple) Password-IC (login)	
	Cert	Use an authentication certificate to access the VPN		
	Password-IC	Use a pre-shared key stored on an IC smart card (recommended if IC smart card is used)		
	Cert-IC	Use a X.509 authentication certificate stored on an IC smart card (recommended if IC smart card is used) (vmm.iccard.enable MUST be 1)		
vpn.vpn PasswordV6	*ASCII string*	Pre-shared key for user authentication (for binary values, enter in hexadecimal, prefixed with “0x”)		
vpn.vpnIdStringV6	*ASCII string*	VPN ID (FQDN or username) for use with vpnPasswordV4		
vpn.vpnCertFileV6		Target PC's certificate file to use for X.509 authentication for VPN		

Setting Name	Possible Values	Description	Default	Omit
		access. File Format: DER		
vpn.vpnCaCertFileV6		VPN (or other party's) CA certificate file for X.509 authentication on the VPN File Format: DER		
vpn.vpnRsaKeyFileV6		Target PC's RSA private key for X.509 authentication on the VPN File Format: DER		
vpn.vpnSpecifyIssuerV6	TRUE	Request that the VPN (or other party) use a certificate signed by the same CA as BitVisor's certificate via IKE Certificate Request payload	FALSE	
	FALSE	Do not specify any CA in the IKE Certificate Request		
vpn.vpnPhase1CryptoV6	DES	Use Data Encryption Standard encryption	3DES	
	3DES	Use Triple DES encryption		
vpn.vpnPhase1HashV6	SHA-1	Security hash function to use SHA-1 is the only possible value	SHA-1	
vpn.vpnPhase1LifeSecondsV6		IPSec SA lifetime (in seconds)	7200	○
vpn.vpnPhase1LifeKilobytesV6	0	Do not specify IPSec SA lifetime data limit	0	○
	>0	IPSec SA lifetime (in kilobytes)		
vpn.vpnWaitPhase2BlankSpanV6		Connection request timeout. The number of seconds to wait after issuing a connection request and declaring a failed attempt		
vpn.vpnPhase2CryptoV6	DES	Use Data Encryption Standard encryption	3DES	
	3DES	Use Triple DES encryption		

Setting Name	Possible Values	Description	Default	Omit
vpn.vpnPhase2 HashV6	SHA-1	Security hash function to use SHA-1 is the only possible value	SHA-1	
vpn.vpnPhase2LifeSecondsV6		IPSec SA lifetime (in seconds)	7200	○
vpn.vpnPhase2LifeKilobytesV6	0	Do not specify IPSec SA lifetime data limit	0	○
	>0	IPSec SA lifetime (in kilobytes)		
vpn.vpnPhase2 StrictIdV6	TRUE	When “TRUE”, update the guest OS virtual IP address (i.e. packet source address) to a valid value before attempting to make an IPSec VPN connection. When FALSE, the address may be a 0-address when a connection attempt is made.	FALSE	○
	FALSE			
vpn.vpnConnect TimeoutV6		Connection request timeout. The number of seconds to wait after issuing a connection request and declaring a failed attempt	5	○
vpn.vpnIdle TimeoutV6		Connection idle timeout. Number of seconds to allow the established VPN connection to be idle before automatically resetting the connection.	300	○
vpn.vpnPingTarget V6	*IPv6 address range*	The address of a remote host within the established VPN. Periodic PINGs will be sent to this host to determine if the VPN is still active. If this entry is omitted, PINGs are not sent		○
vpn.vpnPing IntervalV6		Length of time between PINGs to the host specified by vpn.vpnPingTargetV6	12	○
vpn.vpnPing		Length of PING message	32	○

Setting Name	Possible Values	Description	Default	Omit
MsgSizeV6		to send to vpn.vpnPingTargetV4		

Table 3: VPN IPv6 Settings

B.2 IC Smart Card Settings

This section of the configuration file can be omitted if the IC smart card login procedure is not being used.

Setting Name	Possible Values	Description	Default
idman.crl01File		Certificate revocation list from the CA to be used when verifying X.509 identity certificates (optional)	
idman.pkc01File		CA certificate file used for verifying the X.509 identity certificate stored in the IC smart card	
idman.randomSeed Size	128	Length of the random data used to generate hashes to verify the identity certificate	128
idman.maxPinLen	16	Maximum length of IC smart card PIN	16
idman.minPinLen	16	Minimum length of IC smart card PIN	16
idman.authentication Method	PKI	Authentication method used by IC smart card	PKI

Table 4: IC Smart Card Settings

B.3 Storage Settings

Multiple encryption keys and storage media are supported. They are indexed with *idx* and *num* respectively.

eg.

```
storage.encryptionKey0.place=XXX
storage.encryptionKey1.place=YYY
storage.conf0.type=AAA
storage.conf1.type=BBB
```

The mapping of encryption keys to storage devices is handled with the

`storage.conf`*num*`.keyindex` field. Multiple storage devices may use the same encryption key.

Setting Name	Possible Values	Description	Default
storage.encryption Key*idx*.place	IC USB <*filename*>	Specify the location of the encryption key to be on an IC smart card, USB flash drive, or in the specified file.	<filename> (login-simple) IC (login)
storage.conf*num*.type	ATA ATAPI AHCI AHCI_ATAPI USB ANY	The type of device to be encrypted	
storage.conf*num*. host_id		The host ID of the device to be encrypted. Typically 0 and 1 for the primary and secondary internal hard drives in a PC. -1 : any host (recommended for USB drives)	
storage.conf*num*. device_id		The device ID of the device to be encrypted. Typically 0 and 1 for the master and slave drives in a PC. -1 : any host (recommended for USB drives)	
storage.conf*num*. lba_low		The first segment offset in the encrypted region of the device.	
storage.conf*num*. lba_high		The last segment offset in the encrypted region of the device.	
storage.conf*num*. keyindex	*0 – maximum idx used to index encryptionKey*	The index of the encryption key to be used for the device.	
storage.conf*num*. crypto_name	aes-xts	The name of the encryption to use.	aes-xts
storage.conf*num*. keybits	256	The number of encryption keybits to use	256
storage.conf*num*. extend		TBD	

Table 5: Storage Settings

B.4 Virtual Machine Settings

vmm.f11panic	0 1	Enable(1)/disable(0) BitVisor ability to generate a panic (stop system operation) when the F11 key is pressed on a PS/2 keyboard	0
vmm.f12msg	0 1	Enable(1)/ disable(0) BitVisor ability to write a status message to the output/debug port when the F12 key is pressed on a PS/2 keyboard	0
vmm.auto_reboot	0 1	Reboot (1) /don't reboot (0) the system after BitVisor exits.	0
vmm.shell	0 1	Enable/disable access to the BitVisor command shell when an error occurs	0
vmm.dbgsh	0 1	Enable/disable access to a command shell at any time	0
vmm.status	0 1	TBD	0
vmm.tty_pro1000	0 1	TBD	0
vmm.tty_pro1000_mac_address	*48bit MAC address*	TBD	FF-FF-FF-FF-FF-FF
vmm.driver.ata	0 1	Enable/disable data encryption on ATA drives	0
vmm.driver.usb.uhci	0 1	Enable/disable data encryption on USB 1.1 devices	0
vmm.driver.usb.ehci	0 1	Enable/disable data encryption on USB 2.0 devices	0
vmm.driver.conceal EHCI	0 1	Hide (1)/don't hide(0) USB 2.0 support	0
vmm.driver.conceal 1394	0 1	Prevent(1)/don't prevent(0) guest OS from accessing IEEE 1394 devices on the target PC	0
vmm.driver.conceal PRO1000	0 1	TBD	0

vmm.driver.vpn. PRO100	0	Enable(1)/disable(0) secure connections over Intel PRO 100 NIC on the target PC	0
	1		
vmm.driver.vpn. PRO1000	0	Enable(1)/disable(0) secure connections over Intel PRO 1000 NIC on the target PC	0
	1		
vmm.driver.vpn. RTL8169	0	TBD	0
	1		
vmm.driver.vpn.ve	0	Enable(1)/disable(0) Virtual Ethernet driver (requires 3rd party tool for operation)	0
	1		
vmm.driver. pci_conceal		TBD	
vmm.iccard.enable	0	Enable IC smart card	0
	1		
vmm.iccard.status	0	Enable testing of the IC smart card current status. Used to determine if the card is still connected after bootup.	0
	1		
vmm.boot_active	0	Boot the MBR after BitVisor loads	0
	1	Boot the first active partition once BitVisor loads	

Table 6: Virtual Machine Settings