17. Encryption

Overview

Zabbix supports encrypted communications between Zabbix components using Transport Layer Security (TLS) protocol v.1.2 and 1.3 (depending on the crypto library). Certificate-based and pre-shared key-based encryption is supported.

Encryption can be configured for connections:

- Between Zabbix server, Zabbix proxy, Zabbix agent, zabbix_sender and zabbix_get utilities
- To Zabbix database from Zabbix frontend and server/proxy

Encryption is optional and configurable for individual components:

- Some proxies and agents can be configured to use certificate-based encryption with the server, while others can use pre-shared key-based encryption, and yet others continue with unencrypted communications (as before)
- Server (proxy) can use different encryption configurations for different hosts

Zabbix daemon programs use one listening port for encrypted and unencrypted incoming connections. Adding an encryption does not require opening new ports on firewalls.

Limitations

- Private keys are stored in plain text in files readable by Zabbix components during startup
- Pre-shared keys are entered in Zabbix frontend and stored in Zabbix database in plain text
- Built-in encryption does not protect communications:
    - Between the web server running Zabbix frontend and user web browser
    - Between Zabbix frontend and Zabbix server
- Currently each encrypted connection opens with a full TLS handshake, no session caching and tickets are implemented
- Adding encryption increases the time for item checks and actions, depending on network latency:
    - For example, if packet delay is 100ms then opening a TCP connection and sending unencrypted request takes around 200ms. With encryption about 1000 ms are added for establishing the TLS connection;
    - Timeouts may need to be increased, otherwise some items and actions running remote scripts on agents may work with unencrypted connections, but fail with timeout with encrypted.
- Encryption is not supported by network discovery. Zabbix agent checks performed by network discovery will be unencrypted and if Zabbix agent is configured to reject unencrypted connections such checks will not succeed.

Compiling Zabbix with encryption support

To support encryption Zabbix must be compiled and linked with one of the supported crypto libraries:

- GnuTLS - from version 3.1.18
- OpenSSL - versions 1.0.1, 1.0.2, 1.1.0, 1.1.1
- LibreSSL - tested with versions 2.7.4, 2.8.2:
    - LibreSSL 2.6.x is not supported
    - LibreSSL is supported as a compatible replacement of OpenSSL; the new `tls_*()` LibreSSL-specific API functions are not used. Zabbix components compiled with LibreSSL will not be able to use PSK, only certificates can be used.

The library is selected by specifying the respective option to "configure" script:

- `--with-gnutls[=DIR]`
- `--with-openssl[=DIR]` (also used for LibreSSL)

For example, to configure the sources for server and agent with *OpenSSL* you may use something like:

```
./configure --enable-server --enable-agent --with-mysql --enable-ipv6 --
with-net-snmp --with-libcurl --with-libxml2 --with-openssl
```

Different Zabbix components may be compiled with different crypto libraries (e.g. a server with *OpenSSL*, an agent with *GnuTLS*).

If you plan to use pre-shared keys (PSK), consider using *GnuTLS* or *OpenSSL 1.1.0* (or newer) libraries in Zabbix components using PSKs. *GnuTLS* and *OpenSSL 1.1.0* libraries support PSK ciphersuites with Perfect Forward Secrecy. Older versions of the *OpenSSL* library (1.0.1, 1.0.2c) also support PSKs, but available PSK ciphersuites do not provide Perfect Forward Secrecy.

Connection encryption management

Connections in Zabbix can use:

- no encryption (default)
- RSA certificate-based encryption
- PSK-based encryption

There are two important parameters used to specify encryption between Zabbix components:

- TLSConnect - specifies what encryption to use for outgoing connections (unencrypted, PSK or certificate)
- TLSAccept - specifies what types of connections are allowed for incoming connections (unencrypted, PSK or certificate). One or more values can be specified.

`TLSConnect` is used in the configuration files for Zabbix proxy (in active mode, specifies only connections to server) and Zabbix agent (for active checks). In Zabbix frontend the TLSConnect equivalent is the *Connections to host* field in *Configuration → Hosts → <some host> → Encryption* tab and the *Connections to proxy* field in *Administration → Proxies → <some proxy> → Encryption* tab. If the configured encryption type for connection fails, no other encryption types will be tried.

`TLSAccept` is used in the configuration files for Zabbix proxy (in passive mode, specifies only connections from server) and Zabbix agent (for passive checks). In Zabbix frontend the TLSAccept equivalent is the *Connections from host* field in *Configuration → Hosts → <some host> → Encryption* tab and the *Connections from proxy* field in *Administration → Proxies → <some proxy> → Encryption*

tab.

Normally you configure only one type of encryption for incoming encryptions. But you may want to switch the encryption type, e.g. from unencrypted to certificate-based with minimum downtime and rollback possibility. To achieve this:

- Set `TLSAccept=unencrypted,cert` in the agent configuration file and restart Zabbix agent
- Test connection with zabbix_get to the agent using certificate. If it works, you can reconfigure encryption for that agent in Zabbix frontend in the *Configuration* → *Hosts* → *<some host>* → *Encryption* tab by setting *Connections to host* to "Certificate".
- When server configuration cache gets updated (and proxy configuration is updated if the host is monitored by proxy) then connections to that agent will be encrypted
- If everything works as expected you can set `TLSAccept=cert` in the agent configuration file and restart Zabbix agent. Now the agent will be accepting only encrypted certificate-based connections. Unencrypted and PSK-based connections will be rejected.

In a similar way it works on server and proxy. If in Zabbix frontend in host configuration *Connections from host* is set to "Certificate" then only certificate-based encrypted connections will be accepted from the agent (active checks) and zabbix_sender (trapper items).

Most likely you will configure incoming and outgoing connections to use the same encryption type or no encryption at all. But technically it is possible to configure it asymmetrically, e.g. certificate-based encryption for incoming and PSK-based for outgoing connections.

Encryption configuration for each host is displayed in the Zabbix frontend, in *Configuration* → *Hosts* in the *Agent encryption* column. For example:

Example	Connections to host	Allowed connections from host	Rejected connections from host
NONE	Unencrypted	Unencrypted	Encrypted, certificate and PSK-based encrypted
CERT NONE PSK CERT	Encrypted, certificate-based	Encrypted, certificate-based	Unencrypted and PSK-based encrypted
PSK NONE PSK CERT	Encrypted, PSK-based	Encrypted, PSK-based	Unencrypted and certificate-based encrypted
PSK NONE PSK CERT	Encrypted, PSK-based	Unencrypted and PSK-based encrypted	Certificate-based encrypted
CERT NONE PSK CERT	Encrypted, certificate-based	Unencrypted, PSK or certificate-based encrypted	-

Connections are unencrypted by default. Encryption must be configured for each host and proxy individually.

zabbix_get and zabbix_sender with encryption

See zabbix_get and zabbix_sender manpages for using them with encryption.

Ciphersuites

Ciphersuites by default are configured internally during Zabbix startup and, before Zabbix 4.0.19, 4.4.7, are not user-configurable.

Since Zabbix 4.0.19, 4.4.7 also user-configured ciphersuites are supported for GnuTLS and OpenSSL. Users may configure ciphersuites according to their security policies. Using this feature is optional (built-in default ciphersuites still work).

For crypto libraries compiled with default settings Zabbix built-in rules typically result in the following ciphersuites (in order from higher to lower priority):

Library	Certificate ciphersuites	PSK ciphersuites
GnuTLS 3.1.18	TLS_ECDHE_RSA_AES_128_GCM_SHA256 TLS_ECDHE_RSA_AES_128_CBC_SHA256 TLS_ECDHE_RSA_AES_128_CBC_SHA1 TLS_RSA_AES_128_GCM_SHA256 TLS_RSA_AES_128_CBC_SHA256 TLS_RSA_AES_128_CBC_SHA1	TLS_ECDHE_PSK_AES_128_CBC_SHA256 TLS_ECDHE_PSK_AES_128_CBC_SHA1 TLS_PSK_AES_128_GCM_SHA256 TLS_PSK_AES_128_CBC_SHA256 TLS_PSK_AES_128_CBC_SHA1
OpenSSL 1.0.2c	ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-SHA256 AES128-SHA	PSK-AES128-CBC-SHA
OpenSSL 1.1.0	ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-CCM8 AES128-CCM AES128-SHA256 AES128-SHA	ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA PSK-AES128-GCM-SHA256 PSK-AES128-CCM8 PSK-AES128-CCM PSK-AES128-CBC-SHA256 PSK-AES128-CBC-SHA
OpenSSL 1.1.1d	TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-CCM8 AES128-CCM AES128-SHA256 AES128-SHA	TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA PSK-AES128-GCM-SHA256 PSK-AES128-CCM8 PSK-AES128-CCM PSK-AES128-CBC-SHA256 PSK-AES128-CBC-SHA

User-configured ciphersuites

The built-in ciphersuite selection criteria can be overridden with user-configured ciphersuites.

User-configured ciphersuites is a feature intended for advanced users who understand TLS ciphersuites, their security and consequences of mistakes, and who are comfortable with TLS troubleshooting.

The built-in ciphersuite selection criteria can be overridden using the following parameters:

Override scope	Parameter	Value	Description
Ciphersuite selection for certificates	TLSCipherCert13	Valid OpenSSL 1.1.1 cipher strings for TLS 1.3 protocol (their values are passed to the OpenSSL function SSL_CTX_set_ciphersuites()).	Certificate-based ciphersuite selection criteria for TLS 1.3 Only OpenSSL 1.1.1 or newer.
	TLSCipherCert	Valid OpenSSL cipher strings for TLS 1.2 or valid GnuTLS priority strings. Their values are passed to the SSL_CTX_set_cipher_list() or gnutls_priority_init() functions, respectively.	Certificate-based ciphersuite selection criteria for TLS 1.2/1.3 (GnuTLS), TLS 1.2 (OpenSSL)
Ciphersuite selection for PSK	TLSCipherPSK13	Valid OpenSSL 1.1.1 cipher strings for TLS 1.3 protocol (their values are passed to the OpenSSL function SSL_CTX_set_ciphersuites()).	PSK-based ciphersuite selection criteria for TLS 1.3 Only OpenSSL 1.1.1 or newer.
	TLSCipherPSK	Valid OpenSSL cipher strings for TLS 1.2 or valid GnuTLS priority strings. Their values are passed to the SSL_CTX_set_cipher_list() or gnutls_priority_init() functions, respectively.	PSK-based ciphersuite selection criteria for TLS 1.2/1.3 (GnuTLS), TLS 1.2 (OpenSSL)
Combined ciphersuite list for certificate and PSK	TLSCipherAll13	Valid OpenSSL 1.1.1 cipher strings for TLS 1.3 protocol (their values are passed to the OpenSSL function SSL_CTX_set_ciphersuites()).	Ciphersuite selection criteria for TLS 1.3 Only OpenSSL 1.1.1 or newer.
	TLSCipherAll	Valid OpenSSL cipher strings for TLS 1.2 or valid GnuTLS priority strings. Their values are passed to the SSL_CTX_set_cipher_list() or gnutls_priority_init() functions, respectively.	Ciphersuite selection criteria for TLS 1.2/1.3 (GnuTLS), TLS 1.2 (OpenSSL)

To override the ciphersuite selection in zabbix_get and zabbix_sender utilities - use the command-line parameters:

- `--tls-cipher13`
- `--tls-cipher`

The new parameters are optional. If a parameter is not specified, the internal default value is used. If a parameter is defined it cannot be empty.

If the setting of a TLSCipher* value in the crypto library fails then the server, proxy or agent will not start and an error is logged.

It is important to understand when each parameter is applicable.

Outgoing connections

The simplest case is outgoing connections:

- For outgoing connections with certificate - use TLSCipherCert13 or TLSCipherCert
- For outgoing connections with PSK - use TLSCipherPSK13 and TLSCipherPSK
- In case of zabbix_get and zabbix_sender utilities the command-line parameters `–tls-cipher13` and `–tls-cipher` can be used (encryption is unambiguously specified with a `–tls-connect` parameter)

Incoming connections

It is a bit more complicated with incoming connections because rules are specific for components and configuration.

For Zabbix **agent**:

Agent connection setup	Cipher configuration
TLSConnect=cert	TLSCipherCert, TLSCipherCert13
TLSConnect=psk	TLSCipherPSK, TLSCipherPSK13
TLSAccept=cert	TLSCipherCert, TLSCipherCert13
TLSAccept=psk	TLSCipherPSK, TLSCipherPSK13
TLSAccept=cert,psk	TLSCipherAll, TLSCipherAll13

For Zabbix **server** and **proxy**:

Connection setup	Cipher configuration
Outgoing connections using PSK	TLSCipherPSK, TLSCipherPSK13
Incoming connections using certificates	TLSCipherAll, TLSCipherAll13
Incoming connections using PSK if server has no certificate	TLSCipherPSK, TLSCipherPSK13
Incoming connections using PSK if server has certificate	TLSCipherAll, TLSCipherAll13

Some pattern can be seen in the two tables above:

- TLSCipherAll and TLSCipherAll13 can be specified only if a combined list of certificate- **and** PSK-based ciphersuites is used. There are two cases when it takes place: server (proxy) with a configured certificate (PSK ciphersuites are always configured on server, proxy if crypto library supports PSK), agent configured to accept both certificate- and PSK-based incoming connections
- in other cases TLSCipherCert* and/or TLSCipherPSK* are sufficient

The following tables show the `TLSCipher*` built-in default values. They could be a good starting point for your own custom values.

Parameter	GnuTLS 3.6.12
TLSCipherCert	NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509
TLSCipherPSK	NONE:+VERS-TLS1.2:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL
TLSCipherAll	NONE:+VERS-TLS1.2:+ECDHE-RSA:+RSA:+ECDHE-PSK:+PSK:+AES-128-GCM:+AES-128-CBC:+AEAD:+SHA256:+SHA1:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509

Parameter	OpenSSL 1.1.1d [1]
TLSCipherCert13	
TLSCipherCert	EECDH+aRSA+AES128:RSA+aRSA+AES128
TLSCipherPSK13	TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
TLSCipherPSK	kECDHEPSK+AES128:kPSK+AES128
TLSCipherAll13	
TLSCipherAll	EECDH+aRSA+AES128:RSA+aRSA+AES128:kECDHEPSK+AES128:kPSK+AES128

[1] Default values are different for older OpenSSL versions (1.0.1, 1.0.2, 1.1.0), for LibreSSL and if OpenSSL is compiled without PSK support.

Examples of user-configured ciphersuites

See below the following examples of user-configured ciphersuites:

- Testing cipher strings and allowing only PFS ciphersuites
- Switching from AES128 to AES256

Testing cipher strings and allowing only PFS ciphersuites

To see which ciphersuites have been selected you need to set 'DebugLevel=4' in the configuration file, or use the `-vv` option for zabbix_sender.

Some experimenting with `TLSCipher*` parameters might be necessary before you get the desired ciphersuites. It is inconvenient to restart Zabbix server, proxy or agent multiple times just to tweak `TLSCipher*` parameters. More convenient options are using zabbix_sender or the `openssl` command. Let's show both.

1. Using zabbix_sender.

Let's make a test configuration file, for example /home/zabbix/test.conf, with the syntax of a zabbix_agentd.conf file:

```
Hostname=nonexisting
ServerActive=nonexisting
TLSConnect=cert
TLSCAFile=/home/zabbix/ca.crt
TLSCertFile=/home/zabbix/agent.crt
TLSKeyFile=/home/zabbix/agent.key
TLSPSKIdentity=nonexisting
TLSPSKFile=/home/zabbix/agent.psk
```

You need valid CA and agent certificates and PSK for this example. Adjust certificate and PSK file paths and names for your environment.

If you are not using certificates, but only PSK, you can make a simpler test file:

```
Hostname=nonexisting
ServerActive=nonexisting
TLSConnect=psk
TLSPSKIdentity=nonexisting
TLSPSKFile=/home/zabbix/agentd.psk
```

The selected ciphersuites can be seen by running zabbix_sender (example compiled with OpenSSL 1.1.d):

```
$ zabbix_sender -vv -c /home/zabbix/test.conf -k nonexisting_item -o 1
2>&1 | grep ciphersuites
```

```
  zabbix_sender [41271]: DEBUG: zbx_tls_init_child() certificate
ciphersuites: TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_GCM_SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-CCM8 AES128-CCM AES128-SHA256
AES128-SHA
  zabbix_sender [41271]: DEBUG: zbx_tls_init_child() PSK ciphersuites:
TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-PSK-AES128-CBC-
SHA256 ECDHE-PSK-AES128-CBC-SHA PSK-AES128-GCM-SHA256 PSK-AES128-CCM8 PSK-
AES128-CCM PSK-AES128-CBC-SHA256 PSK-AES128-CBC-SHA
  zabbix_sender [41271]: DEBUG: zbx_tls_init_child() certificate and PSK
ciphersuites: TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_GCM_SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-CCM8 AES128-CCM AES128-SHA256
AES128-SHA ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA PSK-AES128-
GCM-SHA256 PSK-AES128-CCM8 PSK-AES128-CCM PSK-AES128-CBC-SHA256 PSK-AES128-
CBC-SHA
```

Here you see the ciphersuites selected by default. These default values are chosen to ensure interoperability with Zabbix agents running on systems with older OpenSSL versions (from 1.0.1).

With newer systems you can choose to tighten security by allowing only a few ciphersuites, e.g. only ciphersuites with PFS (Perfect Forward Secrecy). Let's try to allow only ciphersuites with PFS using `TLSCipher*` parameters.

The result will not be interoperable with systems using OpenSSL 1.0.1 and 1.0.2, if PSK is used. Certificate-based encryption should work.

Add two lines to the `test.conf` configuration file:

```
  TLSCipherCert=EECDH+aRSA+AES128
  TLSCipherPSK=kECDHEPSK+AES128
```

and test again:

```
  $ zabbix_sender -vv -c /home/zabbix/test.conf -k nonexisting_item -o 1
2>&1 | grep ciphersuites
  zabbix_sender [42892]: DEBUG: zbx_tls_init_child() certificate
ciphersuites: TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_GCM_SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES128-SHA
  zabbix_sender [42892]: DEBUG: zbx_tls_init_child() PSK ciphersuites:
TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256 ECDHE-PSK-AES128-CBC-
SHA256 ECDHE-PSK-AES128-CBC-SHA
  zabbix_sender [42892]: DEBUG: zbx_tls_init_child() certificate and PSK
ciphersuites: TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_GCM_SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-CCM8 AES128-CCM AES128-SHA256
AES128-SHA ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA PSK-AES128-
GCM-SHA256 PSK-AES128-CCM8 PSK-AES128-CCM PSK-AES128-CBC-SHA256 PSK-AES128-
CBC-SHA
```

The "certificate ciphersuites" and "PSK ciphersuites" lists have changed - they are shorter than before, only containing TLS 1.3 ciphersuites and TLS 1.2 ECDHE-* ciphersuites as expected.

2. TLSCipherAll and TLSCipherAll13 cannot be tested with zabbix_sender; they do not affect "certificate and PSK ciphersuites" value shown in the example above. To tweak TLSCipherAll and TLSCipherAll13 you need to experiment with the agent, proxy or server.

So, to allow only PFS ciphersuites you may need to add up to three parameters

```
  TLSCipherCert=EECDH+aRSA+AES128
  TLSCipherPSK=kECDHEPSK+AES128
  TLSCipherAll=EECDH+aRSA+AES128:kECDHEPSK+AES128
```

to zabbix_agentd.conf, zabbix_proxy.conf and zabbix_server_conf if each of them has a configured certificate and agent has also PSK.

If your Zabbix environment uses only PSK-based encryption and no certificates, then only one:

```
  TLSCipherPSK=kECDHEPSK+AES128
```

Now that you understand how it works you can test the ciphersuite selection even outside of Zabbix, with the `openssl` command. Let's test all three `TLSCipher*` parameter values:

```
  $ openssl ciphers EECDH+aRSA+AES128 | sed 's/:/ /g'
  TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256
ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA
  $ openssl ciphers kECDHEPSK+AES128 | sed 's/:/ /g'
  TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256
ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA
  $ openssl ciphers EECDH+aRSA+AES128:kECDHEPSK+AES128 | sed 's/:/ /g'
  TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256
ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA
ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA
```

You may prefer `openssl ciphers` with option `-V` for a more verbose output:

```
  $ openssl ciphers -V EECDH+aRSA+AES128:kECDHEPSK+AES128
            0x13,0x02 - TLS_AES_256_GCM_SHA384  TLSv1.3 Kx=any      Au=any
Enc=AESGCM(256) Mac=AEAD
            0x13,0x03 - TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any
Au=any  Enc=CHACHA20/POLY1305(256) Mac=AEAD
            0x13,0x01 - TLS_AES_128_GCM_SHA256  TLSv1.3 Kx=any      Au=any
Enc=AESGCM(128) Mac=AEAD
            0xC0,0x2F - ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH
Au=RSA  Enc=AESGCM(128) Mac=AEAD
            0xC0,0x27 - ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=RSA
Enc=AES(128)  Mac=SHA256
            0xC0,0x13 - ECDHE-RSA-AES128-SHA    TLSv1 Kx=ECDH     Au=RSA
Enc=AES(128)  Mac=SHA1
            0xC0,0x37 - ECDHE-PSK-AES128-CBC-SHA256 TLSv1 Kx=ECDHEPSK Au=PSK
Enc=AES(128)  Mac=SHA256
```

```
            0xC0,0x35 - ECDHE-PSK-AES128-CBC-SHA TLSv1 Kx=ECDHEPSK Au=PSK
Enc=AES(128)  Mac=SHA1
```

Similarly, you can test the priority strings for GnuTLS:

```
  $ gnutls-cli -l --priority=NONE:+VERS-TLS1.2:+ECDHE-RSA:+AES-128-
GCM:+AES-128-CBC:+AEAD:+SHA256:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509
  Cipher suites for NONE:+VERS-TLS1.2:+ECDHE-RSA:+AES-128-GCM:+AES-128-
CBC:+AEAD:+SHA256:+CURVE-ALL:+COMP-NULL:+SIGN-ALL:+CTYPE-X.509
  TLS_ECDHE_RSA_AES_128_GCM_SHA256                        0xc0, 0x2f
TLS1.2
  TLS_ECDHE_RSA_AES_128_CBC_SHA256                        0xc0, 0x27
TLS1.2
  Protocols: VERS-TLS1.2
  Ciphers: AES-128-GCM, AES-128-CBC
  MACs: AEAD, SHA256
  Key Exchange Algorithms: ECDHE-RSA
  Groups: GROUP-SECP256R1, GROUP-SECP384R1, GROUP-SECP521R1, GROUP-X25519,
GROUP-X448, GROUP-FFDHE2048, GROUP-FFDHE3072, GROUP-FFDHE4096, GROUP-
FFDHE6144, GROUP-FFDHE8192
  PK-signatures: SIGN-RSA-SHA256, SIGN-RSA-PSS-SHA256, SIGN-RSA-PSS-RSAE-
SHA256, SIGN-ECDSA-SHA256, SIGN-ECDSA-SECP256R1-SHA256, SIGN-EdDSA-Ed25519,
SIGN-RSA-SHA384, SIGN-RSA-PSS-SHA384, SIGN-RSA-PSS-RSAE-SHA384, SIGN-ECDSA-
SHA384, SIGN-ECDSA-SECP384R1-SHA384, SIGN-EdDSA-Ed448, SIGN-RSA-SHA512,
SIGN-RSA-PSS-SHA512, SIGN-RSA-PSS-RSAE-SHA512, SIGN-ECDSA-SHA512, SIGN-
ECDSA-SECP521R1-SHA512, SIGN-RSA-SHA1, SIGN-ECDSA-SHA1
```

Switching from AES128 to AES256

Zabbix uses AES128 as the built-in default for data. Let's assume you are using certificates and want to switch to AES256, on OpenSSL 1.1.1.

This can be achieved by adding the respective parameters in `zabbix_server.conf`:

```
  TLSCAFile=/home/zabbix/ca.crt
  TLSCertFile=/home/zabbix/server.crt
  TLSKeyFile=/home/zabbix/server.key
  TLSCipherCert13=TLS_AES_256_GCM_SHA384
  TLSCipherCert=EECDH+aRSA+AES256:-SHA1:-SHA384
  TLSCipherPSK13=TLS_CHACHA20_POLY1305_SHA256
  TLSCipherPSK=kECDHEPSK+AES256:-SHA1
  TLSCipherAll13=TLS_AES_256_GCM_SHA384
  TLSCipherAll=EECDH+aRSA+AES256:-SHA1:-SHA384
```

Although only certificate-related ciphersuites will be used, TLSCipherPSK* parameters are defined as well to avoid their default values which include less secure ciphers for wider interoperability. PSK ciphersuites cannot be completely disabled on server/proxy.

And in `zabbix_agentd.conf`:

```
TLSConnect=cert
TLSAccept=cert
TLSCAFile=/home/zabbix/ca.crt
TLSCertFile=/home/zabbix/agent.crt
TLSKeyFile=/home/zabbix/agent.key
TLSCipherCert13=TLS_AES_256_GCM_SHA384
TLSCipherCert=EECDH+aRSA+AES256:-SHA1:-SHA384
```